Jochen [illegible]z

Maschinencode-Handbuch für den ZX Spectrum

profisoft

Jochen Merz

Maschinencode-Handbuch für den ZX Spectrum

profisoft

ISBN 3-923985-02-9

Herausgeber: Profisoft GmbH, 4500 Osnabrück, Sutthauser Straße 50/52

Umschlaggestaltung: Otto Burzlaff, Osnabrück
Druck und Gesamtherstellung: A & C Welchert GmbH, 4930 Detmold

Vorwort

Die Fachpresse berichtet zunehmend darüber und der Laie staunt und profitiert davon: Junge Leute entpuppen sich als hervorragende Autoren für den nicht gerade einfachen Bereich der Computerwissenschaft. Dank einer unvoreingenommenen Vorgehensweise gelingt ihnen oft, worum sich die Fachautoren meist vergeblich bemühen: Einen in Sprache und Schwierigkeitsgrad leicht verständlichen Text zu verfassen, der dennoch nicht die Erläuterung auch der schwierigsten Zusammenhänge scheut. So auch, wie wir meinen, in diesem Buch des Autoren Jochen Merz, der z. Z. noch die Sekundarstufe II besucht. Sollten Sie dennoch Anlaß zu sachkundiger Kritik haben, sind alle Anregungen willkommen.

Wir wünschen Ihnen viel Spaß und vor allem neue Erkenntnisse zum Sinclair ZX Spectrum.

Osnabrück, im Oktober 1983

Ihr Team von Profisoft

Inhaltsverzeichnis

TEIL I. MASCHINENCODE
TEIL II. BASIC
TEIL III. ANHANG

I. MASCHINENCODE

1. Der Z80-Mikroprozessor 1
1.1. Die Pins 1
1.2. Die Kontrolleinheit 2
1.3. Die Recheneinheit 2
1.4. Das Befehlsregister 2
1.5. Der Programmzähler 2
1.6. Die Hauptregister 2
2. Das Z80-System 5
3. Zahlenformate und Zahlendarstellung 8
3.1. Binäre Form 8
3.2. Hexadezimalzahlen 8
3.3. Zahlendarstellung im Spectrum 9
3.3.1. Integer 9
3.3.2. Fließkommazahlen 9
4. Grundlage der Maschinencodebefehle 11
5. Die Z80-Maschinencodebefehle 12
5.1. Die Flags 12
5.2. NOP 13
5.3. RET 13
5.4. LD 14
5.5. Rechenoperationen 20
5.5.1. INC 20
5.5.2. DEC 21
5.5.3. ADD 22
5.5.4. SUB 23
5.5.5. ADC 24
5.5.6. SBC 24
5.5.7. CP 24
5.6. Logische Operationen 25
5.6.1. AND 25
5.6.2. OR 26
5.6.3. XOR 27
5.7. Sprungbefehle 27
5.7.1. Absolute Sprünge 28
5.7.2. Relative Sprünge 29
5.8. Unterprogramm-Befehle 31
5.8.1. CALL 31
5.8.2. RET 31
5.8.3. RST 32
5.9. Stapeloperationen 32
5.9.1. PUSH 32
5.9.2. POP 32
5.10. Rotations- und Schiebefehle 33
5.11. Einzelbitverarbeitung 35
5.11.1. SET 35
5.11.2. RES 35
5.11.3. BIT 37

5.12. Blockschiebebefehle 37
5.13. Blocksuchbefehle 38
5.14. Transfer-Befehle 38
5.15. Interrupt-Befehle 39
5.16. Sonstige Befehle für den Akkumulator 41
6. Die verschiedenen Aufgaben des ROM 42
7. Die Benutzung des ROM 45
7.1. CLS 45
7.2. Drucken 45
7.3. PLOT 47
7.4. DRAW 48
7.5. CIRCLE 49
7.6. Ändern der Attribute 50
7.7. POINT 51
7.8. SCREEN$ 51
7.9. ATTR 52
7.10. BEEP 52
8. Beispielprogramme 53
8.1. Links-Scroll 53
8.2. Rechtecke 53
9. Abschließend 56

II. BASIC

1. BASIC-Befehle – Nützliches und Wissenswertes 57
2. Optimierung von BASIC-Programmen 59

III. Anhang

A.1 Die Befehlsparameter und Befehlsadressen 61
A.2 Die Literale 63
A.3 Der Speicher 65
A.4 Weitere Z80-Befehle 66
A.5 Fehler im ROM 68
A.6 Umwandlungstabellen von Zahlensystemen 69
A.7 Z80-Befehlssatz (numerisch) 72
A.8 Index-Befehle 76

Quellen 77

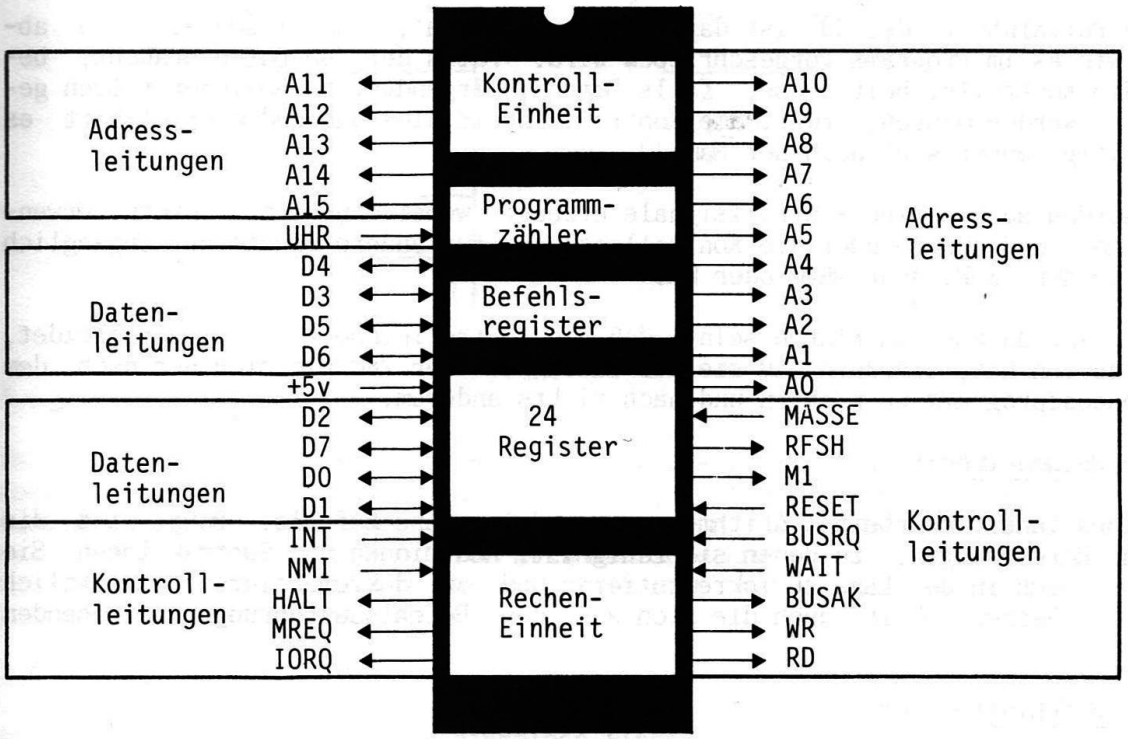
Teil 1

Maschinencode

1. Der Z80 - Mikroprozessor

Der ZX Spectrum besitzt als wichtigsten Chip den ZX80A-Mikroprozessor. Der Z80 ist ein Siliziumchip, der durch 40 Ein- und Ausgabeleitungen, Pins genannt, mit anderen Bauteilen verbunden ist.

Hier ein Diagramm des Z80 mit kurzer Besprechung:



1.1. Zuerst die Funktionen der einzelnen Pins:

A0 - A15 Auf diesen sechzehn Leitungen werden Adressen zum Speicher gebracht. Diese Leitungen werden Adressbus genannt.

D0 - D7 Diese Leitungen stellen den Datenbus dar. Über ihn werden acht Bits vom Speicher in den Prozessor gebracht oder umgekehrt.

UHR Hier wird der Takt in den Z80 geführt. Der Spectrum ist mit 3.5 mHz getaktet, daß heißt, daß alle 0.000000286 Sekunden ein Takt erfolgt.

MASSE Masse.

+5V Die Versorgungsspannung von +5 Volt.

RFSH Normalerweise wird diese Leitung zur Auffrischung vom dynamischem Speicher benutzt; beim Spectrum allerdings dient sie zur Erstellung des Fernsehbildes.

M1 Diese Leitung wird aktiviert, wenn ein Befehl oder Datenbyte aus dem Speicher geholt werden soll.

RESET Diese Leitung wird benutzt, um den Mikroprozessor zu initialisieren.

BUSRQ Es ist möglich , beim Z80 über den Daten- und Adressbus andere Geräte anzusprechen. Dazu dient die BUSRQ-Leitung.

WAIT Diese Leitung wird aktiviert, um dem Z80 anzuzeigen, daß ein Speicher zu langsam ist und noch Zeit für die Datenverarbeitung braucht.

BUSAK Der Mikroprozessor tätigt eine 'Anfrage' durch Aktivieren dieser Leitung.

WR Diese Leitung wird aktiviert, wenn ein Datenbyte vom Z80 in den Speicher gebracht werden soll.

INT Die maskierbare Interrupt-Leitung.

NMI Die nichtmaskierbare Interrupt-Leitung.

HALT Diese Leitung wird aktiviert, wenn ein HALT-Befehl ausgeführt wird.
MREQ Diese Leitung wird immer aktiviert, wenn ein Byte zwischen Prozessor und Speicher transferiert wird.
IORQ Diese Leitung wird aktiviert, wenn ein IN oder OUT-Befehl bearbeitet wird.

1.2. Die Kontrolleinheit

Die Kontrolleinheit des Z80 ist dafür 'veranwortlich', daß im Z80 alles so abläuft, wie es im Programm vorgeschrieben wird. Falls der Z80 Daten braucht, besorgt die Kontrolleinheit diese, falls Daten in irgendwelche Speicherstellen geschrieben werden müssen, führt die Kontrolleinheit dies aus und kontrolliert es gleichzeitig (daher wohl auch der Name!).

Im Z80 werden auch einige Kontrollsignale erzeugt, wobei manche nur intern Verwendung finden und andere über die Kontrolleitungen für andere Bausteine zugänglich gemacht werden, z.B. über MREQ oder WR.

Man muß sich darüber im Klaren sein, daß die Kontrolleinheit nicht entscheidet, was sie zu tun hat, sondern daß sie nur ausführt. Der Z80 hat sich nur nach dem Maschinencodeprogramm zu richten und nach nichts anderem.

1.3. Die Recheneinheit

Die Recheneinheit bearbeitet arithmetische und logische Befehle. Dabei sind die einzigen Berechnungen, zu denen sie fähig ist, Additionen und Subtraktionen. Sie ist aber auch in der Lage zu inkrementieren und zu dekrementieren. Zusätzlich setzt die Recheneinheit auch die sich aus den Befehlsausführungen ergebenden Flags.

1.4. Das Befehlsregister

Wenn der Z80 ein Programm abarbeitet, wird dazu jeder Befehl, der gerade bearbeitet wird, ins Befehlsregister geladen.

1.5. Der Programmzähler

Der Programmzähler ist ein Doppelregister im Z80. Er zeigt entweder auf die Speicherstelle, die gerade abgearbeitet wird, oder in der der Befehl steht, der als nächster in das Befehlsregister geholt werden soll.

Nachdem ein Befehl in das Befehlsregister geladen worden ist, hat die Kontrolleinheit dafür zu sorgen, daß der Programmzähler auf den nächsten Befehl zeigt.

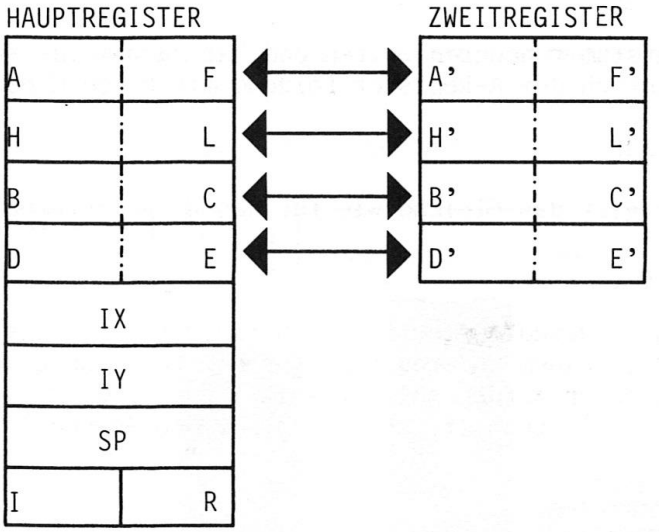
1.6. Die Hauptregister

Es gibt im Z80 noch 24 andere Register, die vom Programmierer benutzt werden können.

Die Namen der 24 Register sind nicht nach logischen Gesichtspunkten gewählt worden; manche Namen stammen noch von Vorgängern des Z80 ab, die noch nicht so kompliziert waren wie er.

Alle 24 Register sind Ein-Byte-Register, wobei die meisten als Doppelregister benutzt werden können.

Die nachstehende Zeichnung zeigt, wie die einzelnen Register kombiniert werden können:



1.6.1. Das A-Register

Dieses Register ist das wichtigste Register des Z80. Es wird auch Akkumulator genannt; dieser Name stammt von der Zeit ab, als es in Mikroprozessoren nur ein einziges Register gab, das einen Wert speichern und verändern konnte. Im Z80 wird das A-Register hauptsächlich für arithmetische und logische Operationen benutzt. Die meisten Einzelregisterfunktionen (wie z.B. Vergleiche) können nur mit dem A-Register durchgeführt werden.

1.6.2. Das F-Register

Das F-Register beinhaltet die einzelnen Flags. Es ist dem Benutzer selbst als Register nicht zugänglich, da nur die einzelnen Flags getestet werden können. Ein Flag ist ein Bit, das in Abhängigkeit von einer Operation entweder ungesetzt (0) oder gesetzt (1) ist. Die Benutzung der Flags wird jedoch noch in einem späteren Kapitel behandelt.

Das Flag-Register enthält 8 Bits, wobei vom Programmierer jedoch nur auf 4 Bits zurückgegriffen werden kann. Diese Bits sind

- das Zero-Flag
- das Carry-Flag
- das Sign-Flag
- das Parity/Overflow-Flag

Dann gibt es noch

- das Half-Carry-Flag
- das Subtract-Flag

die jedoch nur von der Kontrolleinheit benutzt werden.

1.6.3. Das HL-Registerpaar

Der Name stammt auch von früheren Prozessoren ab. Als man zwei Register zu einem Paar zusammenfaßte, benannte man die Register nach ihren Funktionen. Das H-Register enthält daher das high-(höherwertige) Byte und das L-Register das low-(niederwertige) Byte. Viele Operationen, die auf Adressen zugreifen, können nur mit dem HL-Registerpaar durchgeführt werden.

1.6.4. Das BC-Registerpaar

Dieses Registerpaar entstammt neueren Zeiten und hat auch keine spezielle Funktion. Deshalb wurde es einfach dem A-Register folgend mit B und C bezeichnet.

1.6.5. Das DE-Registerpaar

Für dieses Registerpaar gilt das Gleiche wie für das BC-Registerpaar.

1.6.6. Das IX-Registerpaar

Dieses Register wird für indexadressierte Speicherzugriffe benutzt. Im IX-Register steht ein Grundwert und in einem anderen Register die Entfernung vom Grundwert zu der Adresse, die angesprochen werden soll. Das IX-Register wird im Spectrum hauptsächlich für Cassettenbefehle benutzt, so daß es für den Anwender frei ist.

1.6.7. Das IY-Registerpaar

Die Funktionsweise des IY-Registers ist die Gleiche wie beim IX-Register. Nur wird das IY-Register vom Spectrum selbst benutzt und muß grundsätzlich den Wert 23610 (Hexadezimal 5C3A) enthalten.

1.6.8. Der Stackpointer

Der Stackpointer oder Stapelzeiger ist ein Registerpaar, das auf eine Stelle in einem bestimmten Speicherteil, Stack (Stapel) genannt, zeigt. Der Stapel dient zum zeitweiligen Festhalten von Adressen, Rücksprungadressen oder Daten.

Der Stapel des Z80 wird von oben nach unten im Speicher angelegt, das heißt, daß der Z80 seinen Stapel an der für ihn höchstmöglich verfügbaren Adresse aufbaut und jedesmal, wenn ein neues Element auf den Stapel gebracht wird, dieses unter die anderen Elemente gesetzt wird. Dasjenige Element, das zuletzt auf den Stapel gebracht wurde, wird also auch zuerst wieder ausgelesen.

Der Stapelzeiger zeigt immer auf die Adresse der zuletzt auf den Stapel gebrachten Daten; er wird grundsätzlich um zwei erhöht, wenn ein Datenelement ausgelesen wird, und um zwei erniedrigt, wenn etwas Neues auf den Stapel gebracht wird.

Falls Daten für kurze Zeit festgehalten werden sollen, empfiehlt es sich mehr, sie auf den Stapel zu bringen, als Sie in eine Speicherstelle zu schreiben, da das Auf-den-Stapel-bringen wesentlich schneller ist.

1.6.9. Das R-Register

Dieses Register ist das Refresh-Register. Das R-Register ist einfach nur ein Zähler, der jedesmal, wenn Befehle oder Daten aus einer Speicherstelle geholt werden, um eins erhöht wird.

Der Wert im R-Register läuft daher immer von 0 bis 255.

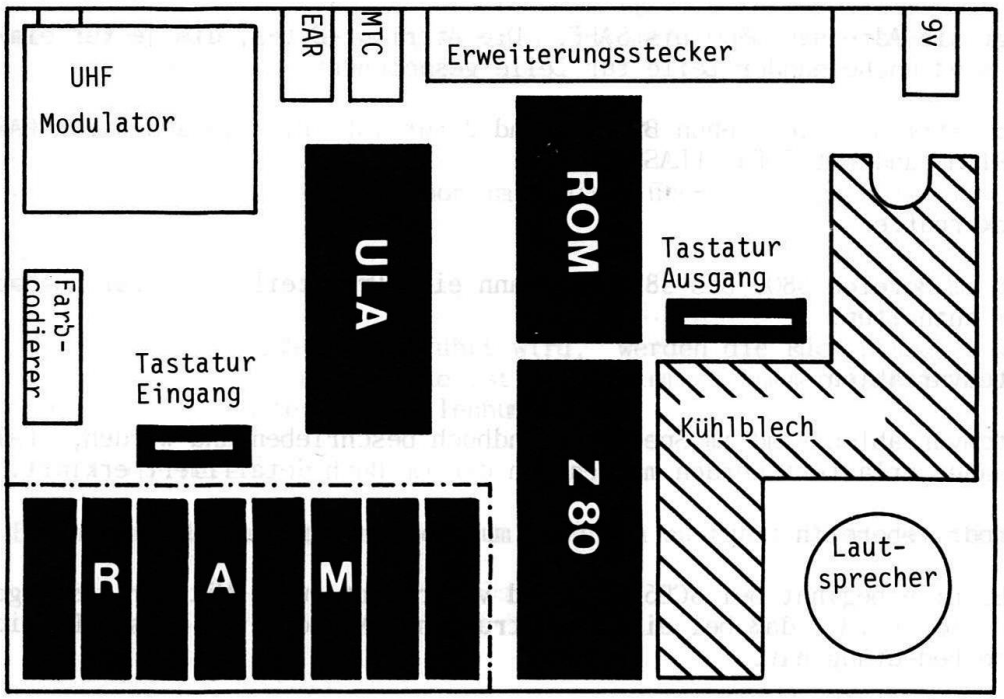
1.6.10. Das I-Register

Dieses Register ist das Interrupt-Register. Wenn der Z80 ein Programm abarbeitet, kann das Programm unterbrochen werden. Solche Interrupts werden meist von externen Geräten erzeugt, wie z.B. von einem Drucker.

2. Das Z80-System

Ein Rechner, der nur aus einer Z80 CPU besteht, wäre undenkbar, da es auch Speicherbausteine geben muß.

Wie die anderen Elemente im Spectrum angeordnet sind, zeigt folgende Abbildung:



Wir besprechen nun alle Hauptbauteile des Spectrums:

Der Z80-Mikroprozessor wurde im vorigen Kapitel schon genauestens besprochen.

Die SINCLAIR ULA
Dieser Chip ist hauptsächlich dafür da, aus dem Bildschirm- und Attributspeicherbereich ein Fernsehsignal zu generieren.

Der Farbcodierer
Dieser Chip wandelt die Farbinformation von der ULA in eine Form um, die der UHF-Modulator verwenden kann.

Der UHF-Modulator
Er generiert das Fernsehbild auf UHF Kanal 36.

Das 16k-ROM
Hierin steht das Festprogramm, das der Z80 normalerweise abarbeitet. Das ROM liegt im Adressbereich von 0000-3FFF; 7k verbraucht das System, 8k der BASIC-Interpreter und 1k der Zeichengenerator.

Das 16 oder 48k-RAM
Das RAM besteht aus Chips zu je 2k-Byte.
Die ersten drei Chips bilden den Bildspeicher (4000 bis 57FF) und der vierte Chip wird unter den Attributen und den Systemvariablen aufgeteilt. Die restlichen Chips sind frei für den Benutzer. Anhang III zeigt die Speicheraufteilung als Diagramm.

Wir sprechen nun die Funktionen der einzelnen RAM-Bereiche genauer durch:

-Der Bildspeicher

Er liegt im Bereich von 4000 bis 57FF und beansprucht wie schon gesagt die ersten drei RAM-Chips, wodurch der Bildschirm in drei Drittel geteilt ist und jeder Chip ein Drittel enthält.

-Der Attributspeicher

Er belegt die Adressen 5800 bis 5AFF. Die Attribut-Bytes, die je für ein Zeichen gelten, sind nacheinander Zeile für Zeile gespeichert.

In einem Attribut-Byte stehen Bit 0,1 und 2 für INK, Bit 3,4 und 5 für PAPER, Bit 6 für BRIGHT und Bit 7 für FLASH.

-Der Druckerpuffer

Er liegt im Bereich 5800 bis 58FF und kann eine Druckzeile in ihrer hochauflösenden Form enthalten.

-Die Systemvariablen

Die Systemvariablen sind im Spectrum-Handbuch beschrieben und werden, falls sie noch genauer erläutert werden müssen, in diesem Buch detailliert erklärt.

-Der Mikrodrivebereich

Dieser Bereich beginnt bei 5CB6 und wird vergrößert, wenn Mikrodrives angeschlossen sind. Das heißt, daß bei einem Spectrum ohne Mikrodrives dieser Bereich keine besondere Bedeutung hat.

-Der BASIC-Bereich

Der BASIC-Bereich besteht aus dem Programm- und dem Variablenbereich. Bei einem normalen Spectrum fängt der Programmbereich bei 5CCB an. In diesem Bereich sind die Zeilen folgendermaßen gespeichert:

- Die ersten zwei Bytes beinhalten die Zeilennummer, wobei das höherwertige Byte zuerst abgespeichert wird.
- Die nächsten zwei Bytes zeigen an, wie lang die Zeile ab dem fünften Byte bis ENTER ist. Diesmal ist das niederwertige Byte zuerst abgespeichert.
- Nun kommt ein Zeilentext beliebiger Länge.
- Das letzte Zeichen der Zeile ist ENTER.

Der Variablenbereich ist auf keine bestimmte Stelle im Speicher festgelegt, da er direkt hinter dem Programmbereich abgelegt wird und es nur auf die Länge des Programms ankommt, wo er liegt. Während der Ausführung eines Programms wird der Variablenspeicher jedoch nicht verschoben. Das Format der Variablen im Speicher wird im Spectrum-Handbuch genau beschrieben.

-Der Editierbereich

Die Größe dieses Bereichs hängt direkt von der Eingabe über die Tastatur ab und liegt direkt hinter dem Variablenspeicher. Es ist dem Benutzer fast unmöglich, diesen Bereich für seine Zwecke zu verändern.

-Der Arbeitsfreiraum

Dieser Freiraum wird für verschiedene Aufgaben gebraucht; z.B. werden hier INPUT-Daten abgelegt. Auch hier kann der Benutzer nichts manipulieren.

-Der Kalkulatorstapel

Auf dem Kalkulatorstapel kann der Z80 alle im Spectrum darstellbaren Zahlen im Fünf-Byte-Format oder ein Fünf- Byte-String-Parameter ablegen. Dieser Stapel ist genauso organisiert wie der Maschinenstapel.

-Der Maschinenstapel

Über den Maschinenstapel wurde schon im Kapitel über den Stackpointer geschrieben.

-Der GOSUB-Stapel

Immer, wenn ein BASIC-GOSUB ausgeführt wird, werden die Rücksprungparameter auf dem Stapel abgelegt. Das erste Byte ist die Statement-Nummer innerhalb der Zeile und die nächsten zwei Bytes die Zeilennummer.

-Der Selbstdefinierbare Zeichensatz

Dieser Bereich wurde bereits im Spectrum Handbuch ausführlich besprochen.

3. Zahlenformate und Zahlendarstellung

3.1. Binäre Form

Das Binärsystem baut auf der Basis 2 auf. Eine Stelle kann also den Wert 0 oder 1 annehmen. Das heißt mit anderen Worten, daß ein Bit zwei Werte annehmen kann. Der Z80 ist in der Lage, 8 oder 16 Bits gleichzeitig zu verarbeiten, wobei 8 Bits zu einem Byte zusammengefaßt werden. Jedes Byte kann nun einen Wert zwischen 0000 0000 und 1111 1111 (dezimal 0 bis 255) annehmen.

Wie kann man nun negative Zanlen darstellen? Kein Register und auch keine Speicherstelle im Z80 kann negative Zahlen darstellen. Man nat sich darauf geeinigt, daß man negative Werte durch das Zahlenkomplement erkennt. Dabei fungiert das 8. Bit von rechts, also Bit 7, als Vorzeichen. Ist es Null, ist die Zahl positiv, ist es eins, negativ. Dadurch ist die Anzahl der Zahlen, die man darstellen möchte, natürlich sehr eingeschränkt. Es sind nur Zahlen zwischen -128 und 127 möglich. Die binären Werte von 0 bis 127 werden weiterhin normal dargestellt, also

```
0000 0000 bin = 0 dez
0000 0001 bin = 1 dez
0000 0010 bin = 2 dez
0000 0011 bin = 3 dez
usw.
```

Die negativen Zahlen sehen dann so aus

```
1111 1111 bin = -1 dez
1111 1110 bin = -2 dez
1111 1101 bin = -3 dez
usw.
```

Ein einfacher Weg, dies zu berechnen, ist:
Man nehme eine negative Zahl, z.B. -13 und forme sie in ihre binäre positive Darstellung um, also 13 dez = 0000 1101 bin. Nun ändere man jedes Bit in sein Gegenteil, aus 0 wird 1, aus 1 wird 0.

00001101 bin ergibt dann 1111 0010.

Nun addiere man 1 dazu: 1111 0010 + 0000 0001 = 1111 0011

Jetzt forme man wieder aufs Dezimalsystem zurück:

1111 0011 bin = 243 dez.

Wie man sieht, ist es recht umständlich, mit Binärzahlen zu arbeiten. Dezimalzahlen sind auch nicht ganz so günstig wie das Zahlensystem, das im nächsten Kapitel beschrieben wird: Das Hexadezimalsystem.

3.2. Hexadezimalzahlen

Diese Form der Zahlendarstellung eignet sich für uns besonders gut. Man kann dadurch mit zwei Zeichen alle 256 möglichen Werte eines Bytes darstellen.

Wenn wir uns wieder die binäre Darstellung einer Zahl anschauen, so teilen wir diese Zahl einfach in der Mitte in je zweimal vier Bits. Diese vier Bits werden Nibble genannt. Jedes Nibble kann jetzt mit den Zeichen 0 - 9 und A - F dargestellt werden:

0000 bin = 0 hex = 0 dez
0001 bin = 1 hex = 1 dez
0010 bin = 2 hex = 2 dez
0011 bin = 3 hex = 3 dez
0100 bin = 4 hex = 4 dez
0101 bin = 5 hex = 5 dez
0110 bin = 6 hex = 6 dez
0111 bin = 7 hex = 7 dez
1000 bin = 8 hex = 8 dez
1001 bin = 9 hex = 9 dez
1010 bin = A hex = 10 dez
1011 bin = B hex = 11 dez
1100 bin = C hex = 12 dez
1101 bin = D hex = 13 dez
1110 bin = E hex = 14 dez
1111 bin = F hex = 15 dez

Dann werden die beiden Hexadezimalzahlen der Nibbles zusammengesetzt, z.B. 1010 1011 bin = AB hex oder 0000 1111 bin = 0F hex.

Eine Zwei-Byte-Zahl muß man nur in vier Nibbles aufteilen, z.B. 1010 1011 1100 1101 bin = ABCD hex.

3.3 Zahlendarstellung im Spectrum

Wie schon vorher erwähnt, stellt der Spectrum intern alle Zahlen im Fünf-Byte-Format dar. Der Spectrum unterscheidet nur zwei Arten von Zahlen:

3.3.1 Integer

Integer sind ganze Zahlen, die im Spectrum Werte von -65535 dez bis 65535 dez annehmen können.

Diese Zahlen werden in folgender Fünf-Byte-Form dargestellt:

- Das erste Byte ist immer 0.
- das zweite Byte fungiert als Vorzeichen; es enthält Null für positive, 255 dez oder FF hex für negative Zahlen.
- Das dritte Byte enthält das niederwertige, das vierte das höherwertige Byte des Betrags des Integers.
- Das fünfte Byte ist auch 0.

Hier ein paar Beispiele:

Zahl: 0 Bytes 1 - 5: 0 0 0 0 0
Zahl: 7777 Bytes 1 - 5: 0 0 97 30 0
Zahl: -3 Bytes 1 - 5: 0 255 3 0 0

3.3.2 Fließkommazahlen

Im Spectrum können Zahlen von 0.3E-39 bis 0.17E39 dargestellt werden. Wenn die Zahl 0 ist, sind alle fünf Bytes Null. Alle anderen Zahlen enthalten als erstes Byte den Exponenten, die restlichen vier die Mantisse.

Gehen wie dies einmal durch:

Nehmen wir die Zahl 1111,2. Um nun die Zahl in Exponent-Form zu bringen, müssen Sie den Dezimalpunkt bis vor die erste Stelle der Zahl schieben, so daß in diesem Beispiel die Zahl .11112 entstehen würde. Der Punkt wurde um vier Stellen verschoben, so das der Exponent 4 ist. Die Mantisse ist .11112 (oder auch 0.1112).

Hier ist dieses Verfahren mit einer Dezimalzahl durchgeführt worden; da der ZX Spectrum aber nur mit Binärzahlen arbeitet, müssen Sie sich das Ganze nun binär übertragen.

Nehmen wir die Binärzahl 1111, also 15 dez. Um das Komma vor die erste Eins zu schieben, muß es um vier Stellen verschoben werden. Nun ist der Exponent 4 und die Mantisse ist 1111 0000 0000 0000 0000 0000 0000 0000. Im Spectrum wird aber noch mehr mit der Zahl gemacht:

Zum Exponenten wird immer 128 addiert. In unserem Falle resultiert daraus 132 (= 4 + 128).

Bei der Mantisse fungiert das erste Byte noch als Vorzeichen. Es wird auf Null gesetzt, wenn die Zahl positiv ist, und auf eins, wenn sie negativ ist. Aus 1111 0000 0000 0000 0000 0000 0000 0000 0000 wird also 0111 0000 0000 0000 0000 0000 0000 0000.

Die Fließkommadarstellung sieht nun dezimal so aus:

132 112 0 0 0

Endgültig sieht das Ergebnis binär so aus:

1000 0100 0111 0000 0000 0000 0000 0000 0000 0000.

Mit allen anderen reellen Zahlen wird ebenso verfahren, nur daß die Umwandlung ein wenig schwieriger als bei Integern ist.

Auch hier ein paar Beispiele:

```
Zahl: 0        Bytes 1 - 5: 0 0 0 0 0
Zahl: 1        Bytes 1 - 5: 129 0 0 0 0
Zahl: 2        Bytes 1 - 5: 130 0 0 0 0
Zahl: 3        Bytes 1 - 5: 130 64 0 0 0
Zahl: 0,5      Bytes 1 - 5: 127 127 255 255 255
Zahl: 1/2      Bytes 1 - 5: 128 0 0 0 0
Zahl: -10      Bytes 1 - 5: 132 160 0 0 0
Zahl: 1,111    Bytes 1 - 5: 129 14 53 63 125
Zahl: -1.111   Bytes 1 - 5: 129 142 53 63 125
```

Der Unterschied zwischen der Darstellung von 0,5 und 1/2 im ZX Spectrum ist sehr interessant.

4. Grundlagen der Maschinencodebefehle

Maschinencodebefehle werden normalerweise in Assembler geschrieben, das heißt, daß jeder Befehl durch ein Mnemonik (Kürzel), das etwas über den Befehl aussagt, für den Programmierer lesbarer gemacht wird.

Da der ZX Spectrum mit der Computersprache BASIC arbeitet, ist es sinnvoll, die Maschinencodestruktur über BASIC zu erklären.

Ein BASIC-Programm besteht aus einer Anzahl von BASIC-Zeilen. Diese Zeilen bestehen aus einer Zeilennummer und Befehlen. Nehmen wir nun an, daß man in jede Zeile nur einen Befehl schreiben könnte. Die Zeilennummern im BASIC-Programm zeigen wie gehabt die Reihenfolge der abzuarbeitenden BASIC-Anweisungen an. Im Maschinencode hat man statt Zeilennummern Adressen, die ebenso in aufsteigender Reihenfolge abgearbeitet werden.

Im BASIC existieren 88 Befehle und Funktionen. Im Maschinencode gibt es über 1100 Maschinencodebefehle, wovon Sie 700 in jedem Assemblerhandbuch finden und der Rest inoffiziell existiert und in Anhang IV zu finden ist.

Jeder dieser Machinencodebefehle kann 1 bis 4 Bytes lang sein, wobei der eigentliche Befehl höchstens 2 Bytes umfassen kann und die anderen beiden Bytes (falls vorhanden) immer Daten sind. Daraus folgt, daß es keinen Z80-Befehl gibt, der mehr als 4 Bytes umfaßt.

Jeder Befehl kann mnemonisch ausgedrückt werden, so daß man sehen kann, was der Befehl bewirken soll. So ist z.B. das Mnemonik LD A, (0284) hexadezimal 7E B4 02, woraus man nur mit sehr viel Übung einen Ladebefehl entschlüsseln kann. Der Befehl LD A,(0284) veranlaßt den Z80 zum Beispiel, den Inhalt der Adresse 0284 (daher die Klammer) in den Akkumulator zu laden (dafür steht LD).

Im Folgenden wird davon ausgegangen, daß der Leser dieses Buches, der Maschinencode lernen möchte, die Computersprache BASIC beherrscht.

Es wird auch jedem empfohlen, nicht auf ein nächstes Kapitel überzugehen, bevor das vorhergehende nicht absolut beherrscht wird.

5. Die Z80-Maschinencodebefehle

In diesem Kapitel werden alle Standart-Z80-Maschinencodebefehle erklärt. Es gibt noch weitere, die allerdings in keinem Assemblerhandbuch zu finden sind und deshalb keine übliche Notation besitzen; sehen Sie bitte dafür in Anhang IV nach.

Um nun die Maschinencodeprogramme eingeben zu können, geben Sie bitte erst folgendes BASIC-Programm ein:

```
 10 CLEAR 29999 : RESTORE : READ H$: LET Z =30000 : LET A=10:  LET B=11: LET C=12:
LET D=13: LET E=14: LET F=15
 20 FOR Q=1 TO LEN H$ STEP 2
 30 POKE Z,VAL H$(Q)*16+VAL H$(Q+1)
 40 LET Z=Z+1
 50 NEXT Q
100 REM (Ab hier die Hex-Daten)
110 DATA ""
```

(Anmerkung: In den spaeter folgenden Maschinencodebeispielprogrammen werden Dezimalzahlen normal und Hexadezimalzahlen mit einem nachgestellten 'H' ausgeschrieben.)

5.1 Die Flags

In diesem Kapitel werden nur die vier Flags besprochen, die vom Programmierer genutzt werden können.

5.1.1 Das Zero-(Null) Flag

Das Zero-Flag ist Bit 6 des F-Registers. (Die acht Bits eines Bytes werden von rechts nach links abgezählt, wobei nicht mit der Zahl Eins, sondern mit Null zu begonnen wird). Das Zero/Flag wird immer dann (auf eins) gesetzt, wenn bei einer Operation der Wert Null herauskommt; anderenfalls wird es auf Null gesetzt. Alle Befehle wie ADC, ADD, AND, CP, DEC, INC, OR, SBC, SUB und XOR, die sich auf ein Ein-Byte-Register beziehen, beeinflussen das Zero-Flag; ebenso die Anweisungen SET, BIT und RES und alle Bitverschiebebefehle, ausgenommen die Ladebefehle.

5.1.2 Das Sign-(Vorzeichen) Flag

Das Sign-Flag ist Bit 7 des F-Registers. Es zeigt an, ob eine Zahl negativ oder positiv ist (Zweierkomplement).

Die Befehle, die das Sign-Flag beeinflussen, sind die gleichen, die für das Zero-Flag gelten.

5.1.3 Das Parity/Overflow-Flag

Dieses Flag ist Bit 2 des F-Registers. Das Parity/Overflow-Flag testet entsprechend dem vorangegangenen Befehl, ob entweder eine gerade oder ungerade Anzahl von gesetzten Bits in dem entsprechenden Register vorliegt (Parity) oder es wird bei Befehlen wie ADD, ADC, SBC und SUB benutzt, um zu testen, ob bei einer Zweierkomplementzahl bei Überschreitung von 0 auch ein richtiges Ergebnis herauskommt.

Nehmen wir an, der Z80 soll 100 dez und 100 dez addieren. Man denkt, es kommen 200 dez heraus. Dies ist allerdings nur bei absoluten Werten der Fall. Das Zweierkomplement von 200 ist aber -54, so daß das P/O-Flag gesetzt wird.

Bei den Befehlen AND , OR und XOR wird getestet, ob das entsprechende Register eine gerade oder ungerade Anzahl von gesetzten Bits enthält. Ist die Anzahl gerade, wird das Flag ungesetzt.

Außerdem wird das P/O-Flag gesetzt, wenn bei INC der Wert 128 oder bei DEC 127 herauskommt.

5.1.4 Das Carry-Flag

Dieses Flag ist Bit 0 des F-Registers.

Es wird gesetzt, wenn durch eine Addition ein zu großes Ergebnis oder durch eine Subtraktion ein negatives Ergebnis resultiert. Das Carry-Flag wird von den Anweisungen ADC, ADD, AND, CP, OR, SBC, SUB und XOR beeinflußt. Ladebefehle verändern es nicht.

Es gibt zwei spezielle Befehle für das Carry-Flag:

SCF Setzt das Carry-Flag.
CCF Komplementiert das Carry-Flag. Das bedeutet, daß, wenn dieses Flag gesetzt ist, es zurückgesetzt wird und umgekehrt.

5.2 NOP

Der No-Operation-Befehl veranlaßt den Z80, ca. 1 Mikrosekunde lang nichts zu tun. Sein Code ist sinnvollerweise 00, da unbenutzte Speicherstellen normalerweise auch den Wert 0 enthalten.

5.3 RET

Der Return-Befehl veranlaßt den Z80, von einem Unterprogramm zu seinem Aufruf zurückzuspringen, der mit CALL (siehe CALL) aufgerufen wurde. CALL und RET im Maschinencode entsprechen in der Funktionsweise den BASIC-Befehlen GO SUB und RETURN. Die BASIC-Anweisung USR ist im Prinzip auch ein Call, so daß mit RET auch ins BASIC zurückgesprungen werden kann, wenn der letzte Stapelwert (die Rücksprungadressen werden z.B. auch auf dem Stapel (Stack) abgelegt) die BASIC-Rücksprungadresse ist, daß heißt, daß das Maschinencodeprogramm mit RANDOMIZE USR aufgerufen worden ist.

Der Hex-Code von RET ist C9.

Hierzu ein Beispiel:

```
ORG 30000
NOP
RET

110 DATA "00C9"
```

Der obere Teil des Listings ist der Source-Code, wie er normalerweise einem Assembler-Programm eingegeben werden kann. Das ORG steht für ORIGINAL und zeigt dem Assembler an, ab welcher Adresse der Code abgelegt werden soll, was in unserem Fall bei 30000 geschieht.

Die untere BASIC-Zeile können Sie nun in das vorher eingegebene BASIC-Programm einfügen und RUN eingeben. Dieses Mini-Maschinencodeprogramm ist nun fertig und liegt ab Adresse 30000 im Speicher. Um es zu starten, benutzen Sie die USR-Funktion. Hinter USR muß die Startadresse des Maschinencodeprogramms stehen. Da USR 30000 aber nicht in dieser Form vom Computer angenommen wird, müssen Sie noch einen Befehl davorsetzen. Der einfachste Befehl ist RANDOMIZE, der in Verbindung mit USR 30000 nur das Maschinencodeprogramm zum Laufen bringt.

Nun besteht noch die Möglichkeit, einen Wert aus dem Maschinencodeprogramm mittels des BC-Registers ins BASIC zu übernehmen. Sie bekommen z.B. durch PRINT USR xxxxx den Wert ausgedruckt, den das BC-Register vor dem Rücksprung beinhaltet. Sie können unter anderem den Wert auch einer Variablen durch LET A=USR xxxxx zuweisen.

Geben sie nun RANDOMIZE USR 30000 ein. Nachdem sie ENTER gedrückt haben, führt das Maschinencodeprogramm einen Leerbefehl aus und springt wieder in BASIC zurück.

5.4 LD

5.4.1 Laden eines Registers mit einer Konstanten

Jedes Register kann mit einer Konstanten geladen werden. Ein-Byte-Konstanten werden im folgenden mit **dd** und Zwei-Byte-Konstanten mit **dddd** bezeichnet. Ein Platzhalter, der für jedes Ein-Byte-Register stehen kann, ist r. Der Platzhalter für die Registerpaare wäre somit rr.

Die möglichen Befehle sind

```
LD A, dd      3E dd
LD B, dd      06 dd
LD C, dd      0E dd
LD D, dd      16 dd
LD E, dd      1E dd
LD H, dd      26 dd
LD L, dd      2E dd
```

Auch hierzu ein Beispielprogramm:

```
ORG 30000
LD B, 1
LD C, 1
RET

110 DATA "06010E01C9"
```

Dieses Programm starten Sie bitte mit PRINT USR 30000, woraufhin auf dem Bildschirm die Zahl 257 erscheint. Wie das?

Zuerst wird B mit 1 geladen. Da BC mit PRINT USR gedruckt wird und B das höherwertige Byte darstellt, muß man den B-Wert mit 256 multiplizieren. 1*256 ergibt also 256.

Da C niederwertig ist, kann der Wert übernommen werden. 256+C=256+1=257.

5.4.2 Laden eines Registers in ein anderes

Jedes der Register A, B, C, D, E, H und L kann in ein anderes dieser Register geladen werden.

Das R- und I-Register kann nur in dem Akkumulator geladen werden.

Daraus ergibt sich auch umgekehrt, daß nur der Akkumulator in I und R geladen werden kann.

	LD A,	LD B,	LD C,	LD D,	LD E,	LD E,	LD H,	LD L,	LD I,	LD R,
A	7F	47	4F	57	5F	67	67	6F	ED 47	ED 4F
B	78	40	48	50	58	60	60	68	-	-
C	79	41	49	51	59	61	61	69	-	-
D	7A	42	4A	52	5A	62	62	6A	-	-
E	7B	43	4B	53	5B	63	63	6B	-	-
H	7C	44	4C	54	5C	64	64	5C	-	-
L	7D	45	4D	55	5D	65	65	6D	-	-
I	ED 57	-	-	-	-	-	-	-	-	-
R	ED 5F	-	-	-	-	-	-	-	-	-

Nun hierzu ein Beispiel. Wir nehmen dafür das vorhergehende Programm: In B wird 1 geladen; da wir den Wert auch in C benötigen, kann B direkt in C geladen werden, so daß das Ergebnis nach PRINT USR 30000 das Gleiche sein müßte.

Probieren Sie:

```
ORG 30000
LD B,1
LD C,B
RET

110 DATA "060148C9"
```

Sie sehen, daß das Ergebnis das gleiche ist; nur ist diese Art der Programmierung eleganter und man spart ein Byte Speicherplatz.

5.4.3 Laden eines Doppelregisters mit einer Konstanten

Die möglichen Befehle sind:

```
LD BC,dddd      01 dd dd
LD DE,dddd      11 dd dd
LD HL,dddd      21 dd dd
LD SP,dddd      31 dd dd
LD IX,dddd      DD 21 dd dd
LD IY,dddd      FD 21 dd dd
```

Doppelregisterladebefehle laden die nächstenfolgenden zwei Bytes als Daten in das entsprechende Doppelregister, und zwar das erste Byte in das niederwertige Register (BC, DE, HL, SP, IX, IY) und das zweite Byte in das höherwertige Register (BC, DE, HL, SP, IX, IY).

Das nachfolgende Beispiel lädt BC mit 1000 dez und springt zurück ins BASIC. Auch dieses Programm wird mit PRINT USR gestartet, damit man das Ergebnis sehen kann.

```
ORG 30000
LD BC,1000
RET

110 DATA "01E803C9"
```

5.4.4 Laden eines Doppelregisters in ein anderes

Von dieser Befehlsart gibt es nur drei Befehle, die sich alle auf das SP-Register beziehen, da alle anderen Registerumladungen auch durch Einzelregisterumladungen ersetzt werden können. Beispiel: LD BC,HL = LD B,H und LD C,L

Die drei möglichen Befehle sind

```
LD SP,HL      F9
LD SP,IX      DD F9
LD SP,IY      FD F9
```

Diese drei Befehle an einem Beispiel zu erklären, dürfte jetzt wohl überflüssig sein.

5.4.5 Laden eines Registers oder Doppelregisters mit dem Inhalt einer Speicheradresse

Der Z80 besitzt viele Befehle, die Daten aus einer Speicherstelle holen und in ein Register bringen. Jeder dieser Befehle muß desnalb zwei Angaben enthalten:

- die Adresse, aus der die Daten geholt werden sollen und
- das Register, in welches die Daten gebracht werden sollen.

Die Klammern um die Adresse bedeuten, daß nicht die Adresse als Wert, sondern ihr Inhalt geladen werden soll. Es wird dabei noch zwischen zwei Adressierungen unterschieden:

5.4.5.1 Absolute Adressierung

Bei den nachfolgenden Befehlen steht die Adresse fest, aus der die Daten geholt werden sollen:

```
LD A,(addr)       3A addr
LD HL,(addr)      2A addr       oder ED 68 addr
LD BC,(addr)      ED 4B addr
LD DE,(addr)      ED 5B addr
LD IX,(addr)      DD 2A addr
LD IY,(addr)      FD 2A addr
LD SP,(addr)      ED 7B addr
```

Auch hier gilt, daß ein einzelnes Byte nur in den Akkumulator geladen werden kann. Bei den anderen Befehlen wird in das niederwertige Register der Inhalt von addr und in das höherwertige der Inhalt von addr + 1 geladen. Selbstverständlich wird auch bei der Adressencodierung (das heißt, bei der Adressangabe) das niederwertige Byte vorangesetzt.

Im Beispiel laden wir den RAMTOP in BC, und wenn das Programm mit PRINT USR 30000 gestartet wird, muß die höchstmöglich verfügbare RAM-Speicherstelle (RAMTOP) ausgegeben werden.

```
ORG 30000
LD BC,(23732)
RET

110 DATA "ED4BB45CC9"
```

Sie können das Ergebnis überprüfen, indem Sie

PRINT PEEK 23732+256xPEEK 23733 ENTER

eingeben.

5.4.5.2 Indirekte Adressierung

Bei der indirekten Adressierung wird der Inhalt aus einer Speicherstelle geholt, deren Adresse in einem Doppelregister festgehalten ist. Dieses Doppelregister kann mit beliebigen Adressen geladen werden, so daß man mit diesen Befehlen den ganzen Speicher adressieren kann. Auch hier ist jede Variation nur mit dem Akkumulator möglich, da alle anderen Register nur mit (HL) geladen werden können.

```
LD A,(HL)       7E
LD A,(BC)       0A
LD A,(DE)       1A
LD B,(HL)       46
LD C,(HL)       4E
LD D,(HL)       56
LD E,(HL)       5E
LD H,(HL)       66
LD L,(HL)       6E
```

POKEn Sie zuerst in Adresse 31000 irgendeinen Wert, worauf Sie dann die nachstehenden Werte eingeben und das MC-Programm mit PRINT USR 3000 starten.

Sehen Sie sich das Beispiel an. In HL wird die Adresse geladen, aus der Daten geholt werden sollen. B wird aud Null gesetzt und der Inhalt der Adresse, die in HL steht, wird in C geladen.

```
ORG 30000
LD HL,31000
LD B,0
LD C,(HL)
RET

110 DATA " 21187906004EC9"
```

5.4.5.3 Index-Adressierung

Die Indexadressierung wird z.B. dazu benutzt, Daten aus Tabellen oder Listen zu lesen. Mit den Indexregistern IX und IY kann man auf Speicherblöcke, die bis zu 256 Bytes groß sein können, zeigen. Das Registerpaar IX kann uneingeschränkt benutzt werden und beim ZX Spectrum muß man vor der Änderung des IY-Registers den maskierbaren Interrupt ausschalten und vor dem Einschalten des Interrupts das Doppelregister IY mit 5C3A laden. Die Basisadresse des Speicherblocks steht im Indexregister; dazu kommt dann noch ein Zweierkomplement-Datenbyte, welches je nach Wert vom Indexregister abgezogen oder dazuaddiert wird. Wenn das Datenbyte 0 ist, zeigt das Indexregister direkt auf die erste Speicherstelle des Blocks.

Man kann sich den Speicherblock folgendermaßen vorstellen:

```
          IX + 7F   ▲
          IX + 7E   |
          IX + 7D   |
          IX + 7C   |
           . . .    |
           . . .    |
           . . .    |
          IX + 02   |
          IX + 01   |
IX   =    IX + 00   |   Adressenzählrichtung
          IX + FF   |
          IX + FE   |
          IX + FD   |
           . . .    |
           . . .    |
           . . .    |
          IX + 81   |
          IX + 80   |
```

Hier die Auflistung der Befehle:

LD A,(IX+dd)	DD 7E dd	LD A, (IY+dd)	FD 7E dd
LD B,(IX+dd)	DD 46 dd	LD A, (IY+dd)	FD 46 dd
LD C,(IX+dd)	DD 4E dd	. . .	
LD D,(IX+dd)	DD 56 dd	. . .	
LD E,(IX+dd)	DD 5E dd		
LD H,(IX+dd)	DD 66 dd		
LD L,(IX+dd)	DD 6E dd		

Man braucht , um IX-Befehle aufs Register zu ändern, nur das DD des Befehlscode in FD ändern.

Wir wollen nun versuchen, das vorherige Beispiel indexadressiert zu lösen. Lassen wir IX auf 30999 dez zeigen, so ist die Differenz +1. POKEn Sie wieder in die Adresse 31000 irgendeinen Wert, geben Sie das nachfolgende Programm ein und starten Sie wieder mit PRINT USR. Es sollte Ihr eingegebener Wert wieder erscheinen.

```
ORG 30000
LD IX,30999
LD B,0
LD C,(IX + 1)
RET

110 DATA "DD2117790600DD4E01C9"
```

5.4.6 Laden einer Speicheradresse mit dem Inhalt eines Registers

Diese Befehle sind sozusagen die 'Umkehrbefehle' der vorhergehenden Adressenladebefehle. Auch hier unterscheidet man wieder zwischen zwei Adressierungsarten:

5.4.6.1 Absolute Adressierung

```
LD (addr),A          32 addr
LD (addr),HL         22 addr  oder ED 63 addr
LD (addr),BC         ED 43 addr
LD (addr),DE         ED 53 addr
LD (addr),IX         DD 22 addr
LD (addr),IY         FD 22 addr
LD (addr),SP         ED 73 addr
```

Es gibt keinen Befehl, mit dem man einen konstanten Wert direkt in eine konstante Adresse laden kann. Die Adressencodierung erfolgt auch hier selbstverständlich nach dem low-high-Prinzip (Niederwertig-Höherwertig).

Geben Sie POKE 31000,22 ENTER und daraufhin das nachstehende Programm ein. Diesmal starten Sie es mit RANDOMIZE USR und PEEKen danach die Adresse 31000 (PRINT PEEK 31000). Es sollte sich der Wert 255 darin befinden.

```
ORG 30000
LD A,255
LD (31000),A
RET

110 DATA "3EFF321879C9"
```

Zuerst wird A mit dem Wert 255 geladen, der dann in die Adresse 31000 gebracht wird.

5.4.6.2 Indirekte Adressierung

```
LD (HL),A            77
LD (BC),A            02
LD (DE),A            12
LD (HL),B            70
LD (HL),C            71
LD (HL),D            72
LD (HL),E            73
LD (HL),H            74
LD (HL),L            75
```

Diese Befehle brauchen nun nicht mehr besonders erklärt werden.

5.4.6.3 Index-Adressierung

```
LD (IX+dd),A         DD 77 dd
LD (IX+dd),B         DD 70 dd
LD (IX+dd),C         DD 71 dd
LD (IX+dd),D         DD 72 dd
LD (IX+dd),E         DD 73 dd
LD (IX+dd),H         DD 74 dd
LD (IX+dd),L         DD 75 dd
LD (IX+dd),dd        DD 36 dd dd
```

Wie auch bei den vorhergenden Index-Befehlen muß bei Verwendung von IY der Code DD in FD geändert werden.

5.4.7 Registeraustausch

Es gibt drei Befehle, die bestimmte Register direkt gegeneinander áustauschen:

```
EX DE,HL          EB
EXX               D9
EX AF,A'F'        08
```

Der Befehl EX DE,HL spricht für sich: DE wird in HL und HL in DE geladen. Der EXX-Befehl tauscht den Hauptregistersatz gegen den Zweitregistersatz mit Ausnahme von A und F, also B, C, D, E, H und L gegen B', C', D', E', H' und L' aus.

Der Befehl EX AF,A'F' hingegen tauscht nur AF gegen A'F' aus. Bitte beachten Sie, daß Sie nie H'L' verändern bzw. den darinstehenden Wert irgendwo zwischenspeichern und später zurückladen, weil sonst ein Rücksprung ins BASIC unmöglich ist.

Auch hierzu ein Beispiel:

```
ORG 30000
LD HL,1000
EX DE,HL
LD B,D
LD C,E
RET

110 DATA "21E803EB424BC9"
```

Wir laden in HL 1000 dez und wollen den Wert durch Umladen in das Registerpaar BC bringen. Wie dies hier gelöst wurde, können Sie sicher selbst verfolgen. Nach PRINT USR 30000 muß auf jeden Fall 1000 auf dem Bildschirm erscheinen.

5.4.8 Austausch eines Doppelregisters gegen den Inhalt einer Speicherstelle

Mit den nachfolgenden drei Befehlen werden die letzten beiden Bytes, die auf den Stapel gebracht wurden, gegen den Inhalt eines Doppelregisters ausgetauscht.

```
EX (SP),HL        E3
EX (SP),IX        DD E3
EX (SP),IY        FD E3
```

5.5 RECHENOPERATIONEN

5.5.1 Inkrementieren

Wenn ein INC-Befehl abgearbeitet wird, wird das entsprechende Register oder Doppelregister um Eins erhöht.

Das Carry-Flag wird durch INC nicht verändert und das Zero-Flag nur bei Ein-Byte-Inkrementierungen.

```
INC A              3C
INC B              04
INC C              0C
INC D              14
INC E              1C
INC H              24
INC L              2C
INC (HL)           34
INC (IX+dd)        DD 34 dd
INC (IY+dd)        FD 34 dd
INC BC             03
INC DE             13
INC HL             23
INC SP             33
INC IX             DD 23
INC IY             FD 23
```

Hierzu ein Beispiel:

```
ORG 30000
LD BC,0100
INC BC
INC BC
INC BC
RET

110 DATA "016400030303C9"
```

Dieses Programm sollte auch wieder mit PRINT gestartet werden. BC wird mit 100 dez geladen und dreimal um Eins erhöht:

100 + 3 x 1 = 103. Probieren Sie's!

5.5.2 Decrementieren

Jeder DEC-Befehl erniedrigt das entsprechende Register oder Doppelregister um Eins.

```
DEC A              3D
DEC B              05
DEC C              0D
DEC D              15
DEC E              1D
DEC H              25
DEC L              2D
DEC (HL)           35
DEC (IX+dd)        DD 35 dd
DEC (IY+DD)        FD 35 dd
DEC BC             0B
DEC DE             1B
DEC HL             2B
DEC SP             3B
DEC IX             DD 2B
DEC IY             FD 2B
```

Hierzu natürlich auch ein Beispiel:

```
ORG 30000
LD BC,0002
DEC BC
DEC BC
DEC BC
RET

110 DATA "0102000B0B0BC9"
```

BC wird mit 2 geladen. Dann wird es dreimal um Eins erniedrigt.

```
BC=2        Erstes DEC      BC=1
BC=1        Zweites DEC     BC=0
BC=0        Drittes DEC     BC=-1=65535
```

Es müßte nach PRINT USR 30000 also der Wert 65535, der ja -1 entspricht, auf dem Bildschirm ausgedruckt werden.

5.5.3 Einfache Addition

Bei der einfachen Addition wird das erste Register oder Doppelregister mit dem zweiten addiert und die Summe ins erste Register oder Doppelregister geladen.

```
Beispiel: LD A,3
          LD B,4
          ADD A,B   entspricht  A = A+B = 7
```

Hier die möglichen Befehle:

```
ADD A,dd            C6 dd
ADD A,A             87
ADD A,B             80
ADD A,C             81
ADD A,D             82
ADD A,E             83
ADD A,H             84
ADD A,L             85
ADD A,(HL)          86

ADD HL,BC           09
ADD HL,DE           19
ADD HL,HL           29
ADD HL,SP           39
ADD IX,IX           DD 29
ADD IX,BC           DD 09
ADD IX,DE           DD 19
ADD IX,SP           DD 39
```

Es braucht eigentlich nicht mehr erwähnt werden, daß, wenn Sie IY statt IX nehmen möchten, nur DD in FD ändern.

Hierzu zwei Programmbeispiele, wobei das erste Ein-Byte-Additionen und das zweite Zwei-Byte-Additionen demonstriert:

Angenommen, in BC steht 10 dez. Dazu soll 20 dez addiert werden. Ein einfaches Programm, daß dies erledigt, ist folgendes:

```
ORG 30000
LD BC,10
LD A,20
ADD A,C
LD C,A
RET

110 DATA "010A003E14814FC9"
```

Natürlich müssen Sie dieses Programm auch wieder mit PRINT USR starten, um das Ergebnis überprüfen zu können.

Nun zur Zwei-Byte-Addition:
Wir möchten 2000 hex zu 4000 hex addieren. Das Registerpaar HL enthält die 4000 hex und BC die 2000 hex. Mit ADD HL,BC wird addiert. Dann wird der Wert nach BC umgeladen und mit PRINT USR muß 24576 dez (6000 hex) angezeigt werden.

```
ORG 30000
LD BC,2000H
LD HL,4000H
ADD HL,BC
LD B,H
LD C,L
RET

110 DATA "01002021004009444DC9"
```

5.5.4 Einfache Subtraktion

Die SUB-Befehle beziehen sich alle auf den Akkumulator, so daß er im Mnemonik nicht mit angeführt wird. Doppelregister können nur mit Carry also mit SBC subtrahiert werden.

Das Register hinter SUB wird vom Akkumulator subtrahiert, wonach das Ergebnis wieder in den Akkumulator kommt.

SUB dd	D6 dd
SUB A	97
SUB B	90
SUB C	91
SUB D	92
SUB E	93
SUB H	94
SUB L	95
SUB (HL)	96
SUB (IX+dd)	DD 96 dd
SUB (IY+dd)	FD 96 dd

Im Beispiel ziehen wir 16 dez von 20 dez ab. Dieses Beispiel braucht nicht näher erklärt zu werden, da es für sich selbst spricht.

```
ORG 30000
LD B,0
LD A,20
SUB 16
LD C,A
RET

110 DATA "06003E14D6104FC9"
```

5.5.5 Addition mit Carry

Die Addition mit dem Carry-Flag verhält sich bei ungesetztem Carry-Flag genauso wie eine normale Addition. Ist das Carry-Flag allerdings gesetzt, wird das Ergebnis einmal inkrementiert.

Hier die möglichen Befehle:

ADC A,dd	CE dd
ADC A,A	8F
ADC A,B	88
ADC A,C	89
ADC A,D	8A
ADC A,E	8B
ADC A,H	8C
ADC A,L	8D
ADC A,(HL)	8E
ADC A,(IX+dd)	DD 8E dd
ADC A,(IY+dd)	FD 8E dd
ADC HL,HL	ED 6A
ADC HL,BC	ED 4A
ADC HL,DE	ED 5A
ADC HL,SP	ED 7A

5.5.6 Subtraktion mit Carry

Die Subtraktion mit dem Carry-Flag verhält sich bei ungesetztem Carry-Flag genauso wie eine normale Subtraktion. Bei gesetztem Carry-Flag wird das Ergebnis einmal dekrementiert.

Die möglichen Befehle:

SBC A,dd	DE dd
SBC A,A	9F
SBC A,B	98
SBC A,C	99
SBC A,D	9A
SBC A,E	9B
SBC A,H	9C
SBC H,L	9D
SBC A,(HL)	9E
SBC A,(IX+dd)	DD 9E dd
SBC A,(IY+dd)	FD 9E dd
SBC HL,HL	ED 62
SBC HL,BC	ED 42
SBC HL,DE	ED 52
SBC HL,SP	ED 72

Die ADC- und SBC-Befehle finden hauptsächlich in der Fließkommaberechnung eine Verwendung, da dadurch leichter Überträge gemacht werden können.

5.5.7 Vergleichsoperationen

Die Vergleichsoperationen erlauben es dem Programmierer, den Inhalt des Akkumulators mit anderen Daten zu vergleichen. Dies geschieht, indem eine Subtraktion simuliert wird und die Flags auf das Ergebnis bezogen gesetzt oder rückgesetzt werden.

Wenn der Akkumulator kleiner als das Vergleichsbyte ist, wird z.B. das Carry-Flag und bei Gleichheit von Akkumulator und Vergleichsbyte das Zero/Flag gesetzt.

Da aber bis zu diesem Zeitpunkt noch keine Befehle, die die Flags abfragen, erklärt worden sind und die Vergleichsoperatoren nur den Zweck der Flagbeeinflussung haben, ist es nicht möglich, hierzu jetzt ein Beispiel zu bringen.

Die Befehle:

CP dd	FE dd
CP A	BF
CP B	B8
CP C	B9
CP D	BA
CP E	BB
CP H	BC
CP L	BD
CP (HL)	BE
CP (IX+dd)	DD BE dd
CP (IY+dd)	FD BE dd

5.6 Logische Operatoren

5.6.1 AND

Der Akkumulator wird mit einem Byte Bit für Bit 'undiert'; daß heißt, daß, wenn zwei Bits gleicher Wertigkeit bzw. gleicher Stelle gesetzt sind, im Ergebnis, das wieder in den Akkumulator geladen wird, das Bit dieser Wertigkeit gesetzt wird. Anderenfalls ist es 0.

Ein Beispiel:

01100010 bin = 62 hex = 98 dez

A N D

11010110 BIN = D6 hex = 214 dez

ergibt

01000010 bin = 42 hex = 66 dez

AND-Operationen setzen das Carry-Flag auf Null.

Hier die Befehlsauflistung:

AND dd	E6 dd
AND A	A7
AND B	A0
AND C	A1
AND D	A2
AND E	A3
AND H	A4
AND L	A5
AND (HL)	A6
AND (IX+dd)	DD A6 dd
AND (IY+dd)	FD A6 dd

Auch hierzu wieder ein Beispiel:

```
ORG 30000
LD B,0
LD A,19
LD C,225
AND C
LD C,A
RET

110 DATA "06003E130EE1A14FC9"
```

Nach PRINT USR 30000 muß 1 herauskommen. Sie können dies überprüfen, indem Sie 19 und 225 mit Hilfe der Umrechnungstabelle im Anhang in das Binärsystem übersetzen und es von Hand undieren.

5.6.2 OR

Durch den Befehl OR wird der Akkumulator mit einem Byte Bit für Bit 'oderiert'; das heißt, daß, wenn mindestens ein Bit der gleichen Wertigkeit bzw. der gleichen Stelle gesetzt ist, auch im Ergebnis das Bit gesetzt wird. Anderenfalls, wenn also das Bit in beiden Bytes Null ist, wird es im Ergebnis auch Null. Hierzu auch gleich ein Beispiel:

```
01100010 bin = 62 hex = 98 dez

      O R

11010110 bin = D6 hex = 214 dez

      ergibt

11110110 bin = F6 hex = 246 dez
```

Die Befehlsauflistung:

OR dd	F6 dd
OR A	B7
OR B	B0
OR C	B1
OR D	B2
OR E	B3
OR H	B4
OR L	B5
OR (HL)	B6
OR (IX+dd)	DD B6 dd
OR (IY+dd)	FD B6 dd

Ein Beispiel (Wenn Sie das vorherige Programm noch im Speicher haben, brauchen Sie nur POKE 30006,177 einzugeben, womit Sie das AND C in OR C ändern):

```
ORG 30000
LD B,0
LD A,19
LD C,225
OR C
LD C,A
RET
```

```
110 DATA "06003E130EE1B14FC9"
```

Das Programm sollte den Wert 243 ausdrucken.

5.6.3 XOR

Durch diesen Befehl wird er Akkumulator mit einem Byte Bit für Bit 'exclusiv oderiert'; das heißt, daß, wenn genau ein Bit der gleichen Wertigkeit gesetzt ist, im Ergebnis das Bit gesetzt wird. Das Beispiel:

```
01100010 bin = 62 hex = 98 dez

     X O R

11010110 bin = D6 hex = 214 dez

     ergibt

10110100 bin = B4 hex = 180 dez
```

Die Befehlsauflistung:

XOR dd	EE dd
XOR A	AF
XOR B	A8
XOR C	A9
XOR D	AA
XOR E	AB
XOR H	AC
XOR L	AD
XOR (HL)	AE
XOR (IX+dd)	DD AE dd
XOR (IY+dd)	FD AE dd

Sie als Leser sollten jetzt selbst in der Lage sein, sich den Befehl in einem Beispielprogramm zu verdeutlichen. Falls Sie im vorhergehenden Programm OR auf XOR ändern, sollte der Wert 242 herauskommen.

Noch ein Tip: Um den Akkumulator auf Null zu setzen, kann man statt LD A,0 besser den Befehl XOR A nehmen; dies ist schneller und spart Speicherplatz!

5.7 Sprungbefehle

Die Sprungbefehle im Maschinencode funktionieren genau so wie das GOTO in BASIC, mit der Ausnahme, daß bei den MC-Sprungbefehlen keine Zeilennummer folgt, ab der das Programm weiter abgearbeitet werden soll, sondern eine Adresse, ab der das Maschinencodeprogramm fortgeführt wird. Diese Adresse wird dann in den Programmzähler geladen. Man kann die Sprungbefehle in zwei Gruppen einteilen:

- die absoluten Sprünge: Mit ihnen kann man zu jeder möglichen Speicheradresse springen.
- die relativen Sprünge: Mit ihnen kann man in einem bestimmten Bereich vorwärts und rückwärts springen.

5.7.1 Absolute Sprünge

Es gibt einen Befehl, bei dem auf jeden Fall zu der nachfolgenden Adresse gesprungen wird:

JP addr C3 addr

Die Adressencodierung ist wie üblich low-high; dies sei hier zum letzten Mal gesagt.

Durch diesen Befehl wird der Programmzähler einfach mit 'addr' geladen.

Zum folgenden Beispiel muß noch etwas erklärt werden. Wenn Sie es sich einmal ansehen, werden Sie sehen, daß hinter dem JP ein Wort steht, nämlich WEITR. Solche Worte, die sozusagen Platzhalter für Adressen darstellen, werden LABEL genannt. Wenn Sie ein Assembler-Programm besitzen, übernimmt es das Ausrechnen der Label und setzt dann die richtige Adresse dafür ein. Das Label muß dann natürlich auch noch in dem MC-Programm dorthingesetzt werden, wo hingesprungen werden soll. Bei dem von mir benutzten Assembler sind die ersten sechs Zeichen jeder Zeile für Label freigelassen, dies ist aber von Assembler zu Assembler unterschiedlich. Sie könnten sich das Beispiel ohne Label so vorstellen:

```
Adresse          Befehl

30000            LD BC,1234
30003            JP 30009
30006            LD BC,7890
30009            RET
```

Nun zum Beispiel: BC wird mit 1234 geladen und es wird direkt zum RET gesprungen, wobei LD BC,7890 ausgelassen wird.

```
ORG 30000
LD BC,1234
JP WEITR
LD BC,7890
WEITR RET

110 DATA "01D204C3397501D21EC9"
```

Es gibt noch drei weitere Befehle, die unbedingt springen. Die Zieladresse muß hierbei in einem Doppelregister stehen.

```
JP (HL)          E9
JP (IX)          DD E9
JP (IY)          FD E9
```

Nehmen wir an, daß in HL 0068 (die Einsprungadresse von CLS) steht; JP (HL) bewirkt dann das gleiche wie JP 0068. Natürlich kann man diese ROM-Unterroutine at 0068 nur mit CALL anspringen; dies sollte aber auch nur ein Beispiel sein.

5.7.1.1 Absolute Sprünge bedingter Art

Wie schon erwähnt, kann man beim Z80 vier Flags testen, und zwar das Carry-, Zero-, Sign- und Parity/Overflow-Flag.

Der Test dieser Flags wird folgendermaßen bezeichnet:

C	(Carry)	wenn das Carry-Flag gesetzt sein soll
NC	(Not Carry)	wenn es nicht gesetzt sein soll
Z	(Zero)	wenn das Zero-Flag gesetzt sein soll
NZ	(Not Zero)	wenn es nicht gesetzt sein soll
M	(Minus)	wenn das Sign-Flag gesetzt sein soll
P	(Plus)	wenn es nicht gesetzt sein soll
PE	(Parity even)	wenn das P/O-Flag gesetzt sein soll
PO	(Parity odd)	wenn es nicht gesetzt sein soll

Jetzt kann jeder Sprung mit einer dieser Bedingungen verknüpft werden: JP Z,addr bedeutet z.B., daß nach addr gesprungen werden soll, wenn das Zero-Flag gesetzt ist. Andernfalls wird das Programm normal nach dem Befehl weiter ausgeführt.

Die möglichen Befehle sind nun:

JP NC,addr	D2 addr
JP C,addr	DA addr
JP NZ,addr	C2 addr
JP Z,addr	CA addr
JP P,addr	F2 addr
JP M,addr	FA addr
JP PO,addr	E2 addr
JP PE,addr	EA addr

Nun können Sie schon unter Verwendung dieser Befehle Schleifen und Ähnliches programmieren: Angenommen, Sie wollen 20x4 in Maschinencode ausrechnen.

Daraus ergibt sich folgende Überlegung: Die Multiplikation ist wiederholte Addition. Man braucht also nur viermal 20 addieren oder 20 mal vier addieren. Das Programm dazu könnte dann folgendermaßen aussehen:

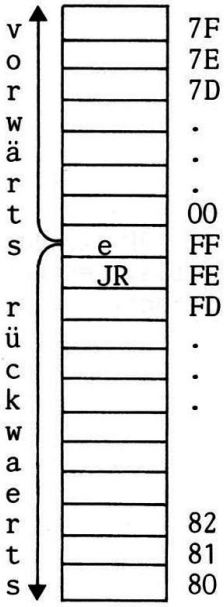
```
ORG 30000
LD B,20
LD C,4
XOR A
MULT ADD A,C
DEC B
JP NZ MULT
LD C,A
RET

110 DATA "06140E04AF8105C235754FC9"
```

(Falls Sie sich nicht mehr erinnern: XOR A setzt den Akkumulator auf Null.) In dem Beispiel dient das Register B als Zähler, der nach jedem Durchlauf um Eins verkleinert wird, so daß die Schleife so oft wiederholt wird, solange es nicht Null ist. C enthält den Wert, der addiert werden soll.

5.7.2 Relative Sprünge

Bei relativen Sprüngen wird nur die Sprungweite vorwärts oder rückwärts angegeben, so daß diese Befehle problemlos mitsamt dem Programm im Speicher verschoben werden können, ohne daß abhängig davon Veränderungen vorgenommen werden müssen. Man spart außerdem noch ein Byte, wenn diese Befehle anstatt von absoluten Sprüngen benutzt werden.

Relative Sprünge können maximal 127 Bytes vorwärts und 128 Bytes rückwärts springen. Diese Sprungweite wird im Nachfolgenden mit dem Platzhalter 'e' gekennzeichnet. Es folgt ein Diagramm, das dies darstellt:



5.7.2.1 Unbedingter Art

Es gibt nur einen Befehl unbedingter Art:

```
JR e                18 e
```

5.7.2.2 DJNZ

Dieser Befehl ist ein automatischer relativer unbedingter Sprungbefehl, der sich auf das B-Register bezieht und 'dekrementiere B und springe falls ungleich Null' bedeutet:

```
DJNZ e              10 e
```

Mit diesem Befehl kann man das Multiplikationsprogramm vereinfachen und kürzen:

```
ORG 30000
LD B,20
LD C,4
XOR A
MULT ADD A,C
DJNZ MULT
LD C,A
RET

110 DATA "06140E04AF8110FD4FC9"
```

5.7.2.3 Bedingter Art

Relative Sprünge bedingter Art können nur das Carry- und Zero-Flag testen:

```
JR NC,e         30 e
JR C,e          38 e
JR NZ,e         20 e
JR Z,e          28 e
```

Es wird nun erklärt, wie man eine Doppelregisterschleife mit Abfrage programmieren kann:

Befehle wie DEC BC, DEC DE und DEC HL setzen bei Erreichen von 0 nicht das Zero-Flag. Angenommen, Sie möchten nur eine Pausenschleife erreichen, die 1000 dez mal durchlaufen wird. Dies könnte man umgefähr so programmieren:

```
LD DE,1000
LOOP  DEC DE
LD A,D      }
OR E        } Ergibt nur 0, wenn D und E Null sind
CP 0
JR NZ LOOP
```

5.8 Unterprogramm-Befehle

Im Maschinencode gibt es drei Arten von Unterprogramm-Befehlen:

CALL, RET UND RST. CALL ist dem BASIC GO SUB gleichzusetzen. RET enspricht dann dem BASIC RETURN und RST ist eine Kurzform von CALL, in der die Adresse gleich mitcodiert ist. Jeder CALL- und RET-Befehl kann die Flags wie die JP-Befehle testen.

5.8.1 CALL

Wenn ein CALL ausgeführt wird, wird die Adresse, nach der wieder zurückgesprungen werden soll, auf den Stapel gebracht. Dann wird die Programmausführung ab der Adresse, zu der durch CALL gesprungen werden soll, fortgesetzt. Die Befehlsauflistung enthält ein unbedingtes CALL. Der Rest bezieht sich auf die vier möglichen Flags.

CALL addr	CD addr
CALL C,addr	DC addr
CALL NC,addr	D4 addr
CALL Z,addr	CC addr
CALL NZ,addr	C4 addr
CALL M,addr	FC addr
CALL P,addr	F4 addr
CALL PE,addr	EC addr
CALL PO,addr	E4 addr

5.8.2 RET

Wenn ein RET ausgeführt wird, wird die zuletzt auf den Stapel gebrachte Adresse in den Programmzähler geladen. Dies ist meistens die Adresse + 3 des letzten CALL-Befehls. Auch hier kann jeder Befehl eines der vier Flags abfragen.

RET	C9
RET C	D8
RET NC	D0
RET Z	C8
RET NZ	C0
RET M	F8
RET P	F0
RET PE	E8
RET PO	E0

5.8.3 RST

Es gibt 8 vordefinierte RST-Befehle, die einen CALL darstellen, in dem die Adresse im Befehlsbyte integriert ist:

```
RST 00      C7          =CALL 0000
RST 08      CF          =CALL 0008
RST 10      D7          =CALL 0010
RST 18      DF          =CALL 0018
RST 20      E7             . . .
RST 28      EF             . . .
RST 30      F7
RST 38      FF
```

Alle diese Befehle springen in das ROM, was Sie im Kapitel 'Die verschiedenen Aufgaben des ROM' nachschauen können.

5.9 Stapeloperationen

Man kann den Stapel auch dazu benutzen, Daten zwischenzuspeichern; dazu braucht man Befehle, die Daten zum Stapel bringen und wieder herunterholen. Zum Auf-den-Stapel-Bringen verwendet man PUSH, da dieser Befehl die zwei Bytes eines Doppelregisters auf den Stapel bringt. POP holt sich die letzten beiden Bytes des Stapel und bringt sie in das gewünschte Doppelregister.

5.9.1 PUSH

```
PUSH AF       F5
PUSH BC       C5
PUSH DE       D5
PUSH HL       E5
PUSH IX       DD E5
PUSH IY       FD E5
```

5.9.2 POP

```
POP AF        F1
POP BC        C1
POP DE        D1
POP HL        E1
POP IX        DD E1
POP IY        FD E1
```

Hierzu wieder ein Beispielprogramm:

```
ORG 30000
LD BC,8888
LD HL,1234
PUSH BC
PUSH HL
POP BC
POP  HL
RET

110 DATA "01B82221D204C5E5C1E1C9"
```

Nach PRINT USR 30000 muß der Wert 1234 ausgegeben werden. BC wird mit 8888 und HL mit 1234 geladen. Danach wird erst BC und dann HL auf den Stapel gebracht.

Der letzte Wert des Stapels ist also 1234. Dieser Wert wird dann in BC und der nächste in HL geladen.

Ganz wichtig: In einer Unterroutine müssen genau so viele Elemente gePOPt wie gePUSHt werden, bevor ein RET ausgeführt wird, da auch die RET-Adresse wie schon erwähnt auf dem Stapel zwischengespeichert wird.

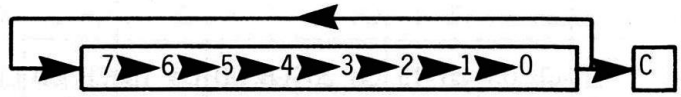
5.10 Rotations- und Schiebebefehle

Es gibt beim Z80 viele Rotations- und Schiebebefehle, um Bits in einem Byte nach rechts oder links zu verschieben. Diese Befehle sind vielseitig verwendbar, da sie z.B., wenn nach links geschoben wird, das Byte verdoppeln und wenn nach rechts geschoben wird, es halbieren. Alle Befehle benutzen das Carry-Flag als Überlaufbit, also als Bit 8 oder als Bit -1.

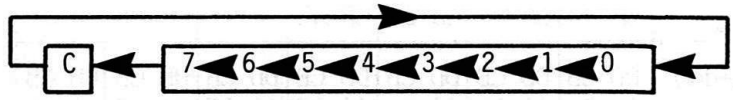
Hier werden die verschiedenen Arten von Rotations- und Schiebebefehlen grafisch veranschaulicht:

RLC / RLCA
Rotation nach links
mit Carry

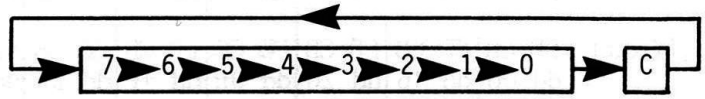
RRC / RRCA
Rotation nach rechts
mit Carry



RL / RLA
Rotation nach links
durch Carry



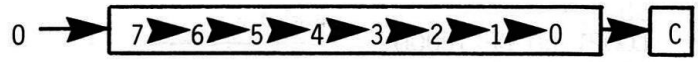
RR / RRA
Rotation nach rechts
durch Carry



SLA
Verschieben nach
links

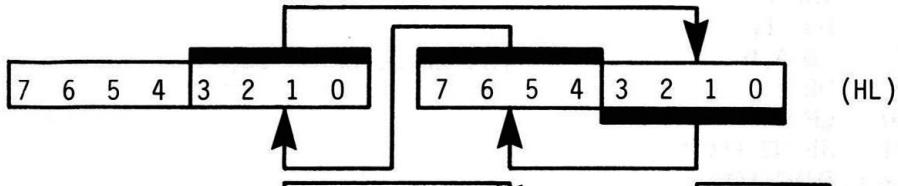
SRA
Verschieben nach
rechts

SRL
Logisches Verschieben
nach rechts



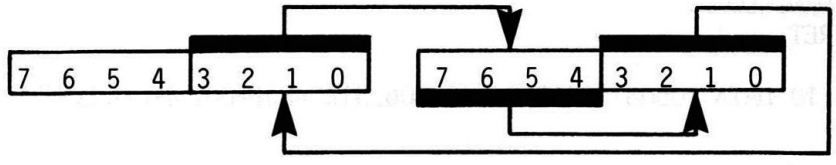
RLD

Akkumulator



RRD

Akkumulator



Die Befehlsauflistung:

Die vier Akkumulatorbefehle:

RLCA	07
RLA	17
RRCA	0F
RRA	1F

	RLC	RL	SLA	RRC	RR	SRA	SRL
A	CB 07	CB 17	CB 27	CB 0F	CB 1F	CB 2F	CB 3F
B	CB 00	CB 10	CB 20	CB 08	CB 18	CB 28	CB 38
C	CB 01	CB 11	CB 21	CB 09	CB 19	CB 29	CB 39
D	CB 02	CB 12	CB 22	CB 0A	CB 1A	CB 2A	CB 3A
E	CB 03	CB 13	CB 23	CB 0B	CB 1B	CB 2B	CB 3B
H	CB 04	CB 14	CB 24	CB 0C	CB 1C	CB 2C	CB 3C
L	CB 05	CB 15	CB 25	CB 0D	CB 1D	CB 2D	CB 3D
(HL)	CB 06	CB 16	CB 26	CB 0E	CB 1E	CB 2E	CB 3E
(IX+dd)	DD CB dd 06	DD CB dd 16	DD CB dd 26	DD CB dd 0E	DD CB dd 1E	DD CB dd 2E	DD CB dd 3E
(IY+dd)	FD CB dd 06	FD CB dd 16	FD CB dd 26	FD CB dd 0E	FD CB dd 1E	FD CB dd 2E	FD CB dd 3E

RRD	ED 67
RLD	ED 6F

Alle Befehle außer RLCA, RLA, RRCA und RRA beeinflussen das Zero-, O/P- und das Sign-Flag.

Zu diesen Befehlen nun ein längeres Beispiel:

```
1     ORG 30000
2     LD B,8
3     LOOP1 LD DE,1800H
4     LD HL,4000H
5     LOOP2 RLC (HL)
6     INC HL
7     DEC DE
8     LD A,D
9     OR E
10    CP 0
11    JR NZ LOOP2
12    DJNZ LOOP1
13    RET

      110 DATA "0608110018210040CB06231B7AB3FE0020F610EEC9"
```

Zu diesem Programm:

Sie sollten vor dem Start mit RAND USR den Bildschirm mit irgendwelchen Zeichen volldrucken. Das Programm läßt die Zeichen alle einmal um sich selbst rotieren.

Um dieses Programm besser erklären zu können, wurde eine Zeilennumerierung der einzelnen Mnemonikkürzel vorgenommen.

Die Funktion der Zeile 1 ist bekannt. In Zeile 2 wird das Register B, das als Zähler fungiert, mit 8 geladen, da jedes Byte 8 mal verschoben werden muß, um in seine Ausgangslage zu gelangen.

Zeile 3 lädt DE mit der Länge und Zeile 4 HL mit dem Anfang des fixen Bildschirmspeichers. Dann rotieren die Bits des Bytes, auf das HL zeigt, um ein Bit nach links (siehe vorhergehende Zeichnung). Der Zeiger HL wird dann auf das nächste Byte gesetzt. Das Registerpaar DE wird in den Zeilen 7 bis 10 bis auf Null heruntergezählt, bis daß alle Zeichen des ganzen Bildschirms um ein Bit verschoben sind. Der DJNZ-Befehl steuert die Anzahl der Verschiebungen (acht Mal).

5.11 Einzelbitverarbeitung

5.11.1 Setzen von Einzelbits

Mit dem SET-Befehl können einzelne Bits in den Registern A, B, C, D, E, H und L in den Speicherstellen (HL), (IX+dd) und (IY+dd) gesetzt werden.

SET	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
A	CB C7	CB CF	CB D7	CB DF	CB E7	CB EF	CB F7	CB FF
B	CB C0	CB C8	CB D0	CB D8	CB E0	CB E8	CB F0	CB F8
C	CB C1	CB C9	CB D1	CB D9	CB E1	CB E9	CB F1	CB F9
D	CB C2	CB CA	CB D2	CB DA	CB E2	CB EA	CB F2	CB FA
E	CB C3	CB CB	CB D3	CB DB	CB E3	CB EB	CB F3	CB FB
H	CB C4	CB CC	CB D4	CB DC	CB E4	CB EC	CB F4	CB FC
L	CB C5	CB CD	CB D5	CB DD	CB E5	CB ED	CB F5	CB FD
(HL)	CB C6	CB CE	CB D6	CB DE	CB E6	CB EE	CB F6	CB FE
(IX+dd)	DD CB dd C6	DD CB dd CE	DD CB dd D6	DD CB dd DE	DD CB dd E6	DD CB dd EE	DD CB dd F6	DD CB dd FE
(IY+dd)	FD CB dd C6	FD CB dd CE	FD CB dd D6	FD CB dd DE	FD CB dd E6	FD CB dd EE	FD CB dd F6	FD CB dd FE

5.11.2 Löschen von Einzelbits

Mit RES können Einzelbits in den oben aufgeführten Registern und Speicherstellen zurückgesetzt werden.

RES	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
A	CB 87	CB 8F	CB 97	CB 9F	CB A7	CB AF	CB B7	CB BF
B	CB 80	CB 88	CB 90	CB 98	CB A0	CB A8	CB B0	CB B8
C	CB 81	CB 89	CB 91	CB 99	CB A1	CB A9	CB B1	CB B9
D	CB 82	CB 8A	CB 92	CB 9A	CB A2	CB AA	CB B2	CB BA
E	CB 83	CB 8B	CB 93	CB 9B	CB A3	CB AB	CB B3	CB BB
H	CB 84	CB 8C	CB 94	CB 9C	CB A4	CB AC	CB B4	CB BC
L	CB 85	CB 8D	CB 95	CB 9D	CB A5	CB AD	CB B5	CB BD
(HL)	CB 86	CB 8E	CB 96	CB 9E	CB A6	CB AE	CB B6	CB BE
(IX+dd)	DD CB dd 86	DD CB dd 8E	DD CB dd 96	DD CB dd 9E	DD CB dd A6	DD CB dd AE	DD CB dd B6	DD CB dd BE
(IY+dd)	FD CB dd 86	FD CB dd 8E	FD CB dd 96	FD CB dd 9E	FD CB dd A6	FD CB dd AE	FD CB dd B6	FD CB dd BE

Zu diesen beiden Befehlen ein Beispiel:

Angenommen, Sie haben auf dem Bildschirm die PAPER-Farbe 7.

Diese Farbe möchten Sie, ohne INK oder Sonstiges zu beeinflussen, in die Farbe rot ändern. Dies geschieht durch Umänderung der Attribute. Das PAPER ist in den Bits 3, 4 und 5 eines Attribut-Bytes gespeichert. Bei PAPER 2 ist Bit 3=0, Bit 4=1 und Bit 5=0.

Nun das Programm dazu:

```
1         ORG 30000
2         LD DE,0300H
3         LD HL,5800H
4   LOOP2 SET 4,(HL)
5         RES 5,(HL)
6         RES 3,(HL)
7         INC HL
8         DEC DE
9         LD  A,D
10        OR E
11        CP 0
12        JR NZ LOOP2
13        RET

          110 DATA "110003210058CBE6CBAECB9E231B7AB3FE0020F2C9"
```

DE fungiert wieder als Zähler; diesmal wird dieses Doppelregister mit der Länge des Attributspeichers geladen. HL zeigt auf den Anfang des Attributspeichers.

In den Zeilen 4 bis 6 wird PAPER 2 in das entsprechende Attribut gebracht; die Zeilen 8 bis 12 kontrollieren die Wiederholung der Schleife, bis alle Attribute geändert sind.

5.11.3 Testen von Einzelbits

Mit dem BIT-Befehl kann bei den üblichen Registern und Speicherstellen getestet werden, ob ein Bit gesetzt ist oder nicht.

Ist das entsprechende Bit 0, wird das Zero-Flag gesetzt, ist es 1, wird das Flag rückgesetzt.

BIT	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
A	CB 47	CB 4F	CB 57	CB 5F	CB 67	CB 6F	CB 77	CB 7F
B	CB 40	CB 48	CB 50	CB 58	CB 60	CB 68	CB 70	CB 78
C	CB 41	CB 49	CB 51	CB 59	CB 61	CB 69	CB 71	CB 79
D	CB 42	CB 4A	CB 52	CB 5A	CB 62	CB 6A	CB 72	CB 7A
E	CB 43	CB 4B	CB 53	CB 5B	CB 63	CB 6B	CB 73	CB 7B
H	CB 44	CB 4C	CB 54	CB 5C	CB 64	CB 6C	CB 74	CB 7C
L	CB 45	CB 4D	CB 55	CB 5D	CB 65	CB 6D	CB 75	CB 7D
(HL)	CB 46	CB 4E	CB 56	CB 5E	CB 66	CB 6E	CB 76	CB 7E
(IX+dd)	DD CB dd 46	DD CB dd 4E	DD CB dd 56	DD CB dd 5E	DD CB dd 66	DD CB dd 6E	DD CB dd 76	DD CB dd 7E
(IY+dd)	FD CB dd 46	FD CB dd 4E	FD CB dd 56	FD CB dd 5E	FD CB dd 66	FD CB dd 6E	FD CB dd 76	FD CB dd 7E

5.12. Blockverschiebebefehle

Es gibt beim Z80 verschiedene Befehle, die es erlauben, Speicherblöcke zu verschieben. Bei diesen Befehlen steht in HL die Startadresse, in DE die Zieladresse und in BC die Länge des zu verschiebenen Blocks.

5.12.1 Nicht-automatische Blockverschiebebefehle

```
LDI         ED A0
LDD         ED A8
```

Bei diesen Befehlen wird ein einzelnes Byte von (HL) nach (DE) gebracht und das BC-Doppelregister decrementiert. Danach werden die HL- und DE-Doppelregister bei

```
LDI     inkrementiert und bei
LDD     dekrementiert.
```

5.12.2 Automatische Blockverschiebebefehle

```
LDIR        ED B0
LDDR        ED B8
```

Diese Befehle verhalten sich genau so wie die nicht-automatischen Blockverschiebebefehle mit der Erweiterung, daß der Einzelbefehl so oft wiederholt wird, bis BC Null ist.

Das R steht daher für REPEAT. Ein Beispiel hierfür ist das Programm des Seitwärts-Scroll im Beispiel-Anhang.

5.13 Blocksuchbefehle

Auch hier gibt es automatische und nicht-automatische Befehle. Bei diesen Befehlen muß im Akkumulator der Wert stehen, nach dem ab (HL) gesucht werden soll. BC ist die Länge des Bereichs, in dem ab (HL) gesucht werden soll.

5.13.1 Nicht-automatische Blocksuchbefehle

CPI	ED A1
CPD	ED A9

Es wird (HL) mit A verglichen und das Zero-Flag ensprechend gesetzt (wie bei CP). Dann wird BC decrementiert und HL bei

CPI inkrementiert und bei
CPD dekrementiert.

Ist BC bei Null angekommen, wird das P/O-Flag rückgesetzt.

5.13.2 Automatische Blocksuchbefehle

CPIR	ED B1
CPDR	ED B9

Bei CPIR wird theoretisch so lange CPI ausgeführt, bis BC Null ist (dann werden das Zero-, Sign- und P/O-Flag zurückgesetzt) oder A gefunden ist (dann wird das Zero-Flag gesetzt und HL enthält die nachfolgende Adresse des gefundenen Bytes).

Die Funktionsweise des Befehls CPDR ist, angesehen davon, daß HL dekrementiert wird, der von CPIR gleich.

5.14 Transfer-Befehle

Es gibt zwei Befehlsarten, mit denen man externe Einrichtungen wie den Drucker, die Tastatur oder den Cassettenrecorder ansprechen kann. Sie kennen sie vom BASIC her: IN und OUT.

Mit OUT können Sie Daten zu irgendwelchen Geräten schicken und mit IN von ihnen Daten empfangen. Wenn ein OUT abgearbeitet wird, wird auf den Datenbus der Inhalt eines bestimmten Registers geschickt; bei IN wird ein Byte vom Datenbus geholt und in ein bestimmtes Register geladen. Man benötigt aber noch eine bestimmte Adresse, um ein Gerät anzusprechen, nämlich die PORT-Adresse. Diese PORT-Adresse wird bei IN und OUT auf den Datenbus gelegt, um anzuzeigen, womit kommuniziert wird.

Die PORT-Adressen im Machinencode sind die gleichen wie in BASIC.

Die möglichen Befehle:

		Ein- bzw.Ausgaberegister	Höherwert. PORT-Adresse	Niederwert. PORT-Adresse
IN A,(dd)	DB dd	A	A	dd
IN A,(C)	ED 78	A	B	C
IN B,(C)	ED 40	B	B	C
IN C,(C)	ED 48	C	B	C
IN D,(C)	ED 50	D	B	C
IN E,(C)	ED 58	E	B	C
IN H,(C)	ED 60	H	B	C
IN L,(C)	ED 68	L	B	C
OUT (dd),A	D3 dd	A	A	dd
OUT (C),A	ED 79	A	B	C
OUT (C),B	ED 41	B	B	C
OUT (C),C	ED 49	C	B	C
OUT (C),D	ED 51	D	B	C
OUT (C),E	ED 59	E	B	C
OUT (C),H	ED 61	H	B	C
OUT (C),L	ED 69	L	B	C

Als Beispiel hierzu sei auf das Ändern von BORDER in der ROM-Benutzung hingewiesen (Siehe Kapitel 7).

5.14.1 Block-Transfer-Befehle

Diese Befehle werden hier nur ganz kurz aufgelistet, da sie fast ausschließlich im Diskettenbetrieb verwendet werden.

INI	ED A2
INIR	ED B2
IND	ED AA
INDR	ED BA
OUTI	ED A3
OTIR	ED B3
OUTD	ED AB
OTDR	ED 8B

Auch hier steht am Anfang entweder IN oder OUT (OT), danach I für Inkrement oder D für Dekrement und eventuell noch R für Repeat (automatische Befehle).

5.15 Interrupt-Befehle

Der Z80 kann in seiner Programmabarbeitung 'unterbrochen' werden. Das heißt, daß er durch einen Interrupt sein Programm abbricht und an einer Interrupt-Routine (bis zum RET) weiterarbeitet, um danach das normale Programm dort fortzusetzen, wo unterbrochen worden ist.

Es gibt beim Z80 zwei Leitungen, die im Spectrum einen Interrupt bewirken: Die NMI- und die INT-Leitung.

Der NMI ist vom Benutzer nicht zu erreichen und braucht deshalb hier auch nicht erklärt zu werden.

Nun zum INT: Wenn die INT-Leitung aktiviert wird, hört der Z80 mit dem Abarbeiten seines Programms auf, speichert den PC auf dem Stapel ab und fährt je nach Interrupt-Modus mit einem Unterprogramm fort.

Wenn der Z80 eingeschaltet wird, bleibt der maskierbare Interrupt (INT) so lange ausgeschaltet, bis er mit EI aktiviert wird. Im Spectrum dient der INT zur Tastaturabfrage und zur Aktualisierung der Real-Time-Clock. Wie schon erwähnt wird mit

```
EI                    FB
```

der INT abgeschaltet. Man kann ihn mit

```
DI                    F3
```

wieder aktivieren. Bei den Cassettenbefehlen des Spectrums wird der maskierbare Interrupt so gehandhabt, daß die Real-Time-Clock dabei nicht aktualisiert wird. Für den maskierbaren Interrupt gibt es drei verschiedene Modi. Die Modi können durch folgende Befehle gewählt werden.

```
IM 0      ED 46
IM 1      ED 56
IM 2      ED 5E
```

Modus 0 bewirkt, daß, wenn ein Interrupt vorliegt, ein externes Gerät ein Byte, das anzeigt, welche RST-Adresse in den PC geladen werden soll, auf den Datenbus gibt. Modus 0 wird beim Spectrum nicht benutzt.

Modus 1 führt RST 38 aus.

Modus 2 wird normalerweise auch nicht im Spectrum verwendet, kann aber benutzt werden, da hier eine Adresse errechnet wird, die sich aus dem I-Register und einem Byte vom Datenbus errechnet und die eine Adresse in einer Tabelle im Speicher angibt und damit aufzeigt, wohin gesprungen werden soll.

Normalerweise wird mit einem RET von einer Interrupt-Routine zurückgesprungen; es gibt aber noch zwei spezielle Interrupt-Rücksprung-Befehle:

```
RETI      ED 4D
RETN      ED 45
```

Mit RETI wird wie mit RET zurückgesprungen und zusätzlich dem Gerät, welches den INT anmeldete, mitgeteilt, daß der Interrupt beendet ist.

RETN hat die gleiche Arbeitsweise wie RETI, wird jedoch im Zusammenhang mit dem nicht-maskierbaren Interrupt benutzt.

Es gibt einen Befehl, der auf eines Interrupt wartet und so lange NOPs ausführt. bis ein Interrupt eintrifft, nämlich

```
HALT    76
```

Sie können dies mit folgendem Programm ausprobieren:

```
ORG 30000
HALT
RET

110 DATA "76C9"
```

Starten Sie es mit RAND USR, wonach der Spectrum direkt wieder 'zurückkommen' müßte, da intern jede 1/50 Sekunde ein INT vorkommt.

5.16 Sonstige Befehle für den Akkumulator

Mit NEG erhält man das Zweierkomplement des Akkumulators.

```
NEG     ED 44
```

CPL komplementiert den Akkumulator, was bedeutet, daß dieser Befehl jedes gesetzte Bit in ein ungesetztes ändert und umgekehrt.

```
CPL     2F
```

Beispiel:

10010101 bin = 149 dez

C P L
ergibt

01101010 BIN = 106 DEZ

Der letzte Befehl wäre:

```
DAA     27
```

Dieser Befehl bringt den Akkumulator in ein dezimales Format, so daß die Bits 3 bis 0 die Einer-Stelle und die Bits 7 bis 4 die Zehner-Stelle darstellen. Jeder Nibble kann nun Werte von 0 (0000 bin) bis 9 (1001 bin) annehmen. Auch hierzu ein Beispiel:

00100111 bin = 39 dez

D A A
ergibt

0011 1001
3 9

6. Die verschiedenen Aufgaben des ROM

0000 - 0007 Der absolute Start. Der maskierte Interrupt wird ausgeschaltet, A wird auf Null gesetzt und DE wird mit FFFF, der höchstmöglichen RAMTOP-Adresse, geladen. Danach wird nach 11CB gesprungen.

0008 - 000F Die Fehlerausgaberoutine, die den "ERROR-REPORT" am unteren Bildschirmrand ausgibt. Hier wird auch der Prozessorstapel gelöscht.

0010 - 0017 Diese Routine springt nach 15F2, wo das Zeichen, dessen Code im Akkumulator steht, auf den Bildschirm gedruckt wird.

0018 - 0027 Holt das nächste Zeichen, auf das CH-ADD zeigt.

0028 - 002F Springt zum Kalkulator bei 335B.

0030 - 0037 Springt nach 169E, wo BC Leerräume in den Arbeitsspeicher gebracht werden.

0038 - 0065 Diese Routine wird durch den maskierbaren Interrupt angesprungen, um die Echtzeituhr zu stellen.

0066 - 0094 Diese Routine wird durch den nichtmaskierbaren Interrupt angesprungen. Wenn 5CB0 auf Null steht, wird die Funktion NEW ausgeführt.

0095 - 0204 Die Befehls- und Funktionsworttabelle. Hier sind die ausgeschriebenen "TOKEN" verzeichnet.

0205 - 028D Die sechs Tastaturtabellen für je einen CURSOR-Modus.

028E - 02BE Die Tastaturroutinen. Hier wird die Auto-Repeat-Funktion getestet, die Tastaturwerte erkannt und der ASCII-CODE in LAST-K gespeichert.

03B5 - 046D Die BEEP-Routine (Genaueres im Kapitel über das Benutzen von ROM-Routinen).

04AA - 04C1 Fehlerbehandlung.

04C2 - 09F3 Die LOAD-, SAVE-, VERIFY- UND MERGE-Routinen:

- 04C2 - 053E Diese Routine gibt DE Bytes ab IX auf den Recorder.
- 053F - 0555 Ende von LOAD und SAVE.
- 0556 - 0604 Diese Routine lädt oder vergleicht DE Bytes ab IX vom Recorder. Das Carry-Flag wird zum Laden gesetzt und für VERIFY auf Null gesetzt.
- 0605 - 075F Diese Routine erstellt den "Programmkopf" beim Abspeichern.
- 0760 - 096F Diese Routine steuert die anderen Unterroutinen.
- 0970 - 09F3 Diese Routine bereitet das SAVEn durch Drucken von "Start the tape and press any key" (09A1 - 09F3) vor und wartet auf einen Tastendruck. Dann wird der "Kopf" und nach einer Sekunde die Daten abgespeichert.

09F4 - 0D4C Die Druckroutinen. Bit 1 von FLAGS ist auf Null, wenn auf dem Bildschirm ausgegeben wird und entsprechend für den Drucker auf Eins.

- 09F4 - 0A10 Hier werden Kontrollzeichen von druckbaren Zeichen unterschieden.
- 0A11 - 0ADB Hier werden die Kontrollzeichen bearbeitet.
- 0ADC - 0B02 Hier werden die Druckpositionen gespeichert.
- 0B03 - 0B23 Hier werden sie abgerufen.
- 0B24 - 0BDA Hier werden die Zeichen gedruckt. Beim Einsprung enthält HL die Pixel-Adresse, wo das Zeichen hingedruckt werden soll, BC enthält Zeile und Spalte und A den Zeichencode.
- 0BDB - 0C09 Hier werden die Attribute zum gedruckten Zeichen gesetzt.
- 0C0A - 0C54 Diese Routine druckt die "TOKEN".
- 0C55 - 0D4C Diese Routine testet, ob "scroll?" nötig ist. Fall ja, wird es gedruckt und auf einen Tastendruck gewartet.

0D4D - 0D6A Diese Routine setzt die zeitweiligen Attribute.

0D6B - 0EAB Die CLS-Routine (s. ROM-Benutzung)

0EAC - 0F2B Die Drucker-Routinen:
0EAB - 0ECC COPY
0ECD - 0EF3 Der Druckerpuffer wird ausgedruckt.
0EF4 - 0F2B Die Druck-Steuer-Routine.

0F2C - 10A7 Der Editor. Hier wird der Edit-Bereich konstruiert und/oder verändert. Bei korrektem Tastendruck wird hier auch der BEEP-Laut erzeugt.
10A8 - 11B6 Die Tastatureingabe-Routine. Hier wird auf GRAPHICS, SYMBOL SHIFT etc. reagiert.
11B7 - 11CA Hier wird NEW ausgeführt.
11CB - 12A1 Die Initialisierungsroutine. Der Akkumulator enthält Null bei totaler Neuinitialisierung und FF bei NEW.
11CB - 11CF BORDER weiß.
11D0 - 11D9 I wird mit 3F geladen.
11DA - 11EF RAM wird von FFFF abwärts getestet.
11F0 - 11FF wird bei NEW übersprungen und initialisiert P-RAMT, RASP, PIP und UDG.
1200 - 1218 Der Selbstdefinierbare Zeichensatz wird mit A bis U geladen.
1219 - 1234 Der Prozessorstapel wird aufgebaut, IM 1 gewählt und IY auf 5C3A gesetzt. Ab jetzt wird der maskierbare Interrupt und damit die Echtzeituhr wirksam.
1235 - 1243 Die CHANNEL-Informationen werden aufbereitet.
1244 - 1285 Einige Systemvariablen werden initialisiert.
1286 - 12A1 Der Druckpuffer und der Bildschirm werden gelöscht und das Sinclair-Copyright wird gedruckt.

12A2 - 15AE Die Hauptroutine.
12A2 - 12E1 Hier kann eine BASIC-Zeile eingegeben werden, die geprüft und ins Listing übernommen wird.
12E2 - 1302 Hier wird eine BASIC-Zeile ausgeführt.
1303 - 1390 Ein Report wird ausgegeben und an den Anfang der Hauptroutine zurückgesprungen.
1391 - 1554 Tabelle der ausgeschriebenen Fehlermeldungen.
1555 - 15AE Hier wird eine Zeile aus dem Editierbereich ins Programm übernommen.

15AF - 1651 Hier werden die INPUT - OUTPUT - Kanäle benutzt.
1562 - 1654 Ein Leerzeichen wird in den Arbeits- oder Editierbereich gebracht.
1655 - 1663 In BC steht, wieviel Platz nach HL gemacht werden soll.
1664 - 168E Die Zeiger VARS bis STKEND werden auf den neuesten Stand gebracht.
168F - 1690 Zu einer Adresse wird die entsprechende Zeilennummer gefunden.
169E - 16AF Eine bestimmte Anzahl von Bytes wird im Arbeitsspeicher bereitgestellt.
16B0 - 16BE Löscht die Eingabezeile.
16BF - 16C4 Löscht den Arbeitsbereich.
16C5 - 16DA Löscht den Kalkulatorstapel.
16DB - 16E3 Verschiedene niederwertige Aufgaben.
16E4 - 1792 OPEN und CLOSE.
1793 - 1794 Im normalen Spectrum führt der Gebrauch von FORMAT, ERASE, MOVE und CAT zur Fehlermeldung INVALID STREAM.
1795 - 1A47 LIST.
1795 - 186D Druckt eine BASIC-Zeile.
Zuerst wird die Zeilennummer gedruckt, danach, falls nötig, der Zeilenzeiger und zum Schluß der Zeilentext.
196E - 19B7 Diese Routine findet zu einer Zeilennummer die entsprechende Adresse.
19B8 - 19E4 Die nächste BASIC-Zeile wird gesucht.
19E5 - 19FA Daten-Verschiebe-Routine.

1A1B - 1A47 Die Zahl in BC wird als Zeilennummer gedruckt (Nur 0 - 9999; die Routine ist aber trotzdem sehr gut zu benutzen!).
1A48 - 1B16 Die Befehlstabellen. Die erste ist eine Differenztabelle, welche auf die Parametertabelle zeigt (siehe Anhang I).

1B17 - 1C00 Die Kontrollroutine des Interpreters.
1C01 - 1C0C Die Befehlsmöglichkeitendifferenz.
1C0D - 1CDD Die Befehlsmöglichkeitenroutine.
1CDE - 24FA Die Befehlsausführungen.
24FB - 28B1 Die Funktionsausführungen.
24FB - 2534 Die Hauptschleife der Funktionsausführung.
2535 - 257F SCREEN$ (s.Benutzung des ROMs)
25B0 - 25F7 ATTR "
25F8 - 2626 RND "
2627 - 2633 PI "
2634 - 2755 INKEY$ "
2756 - 27BC Kalkulator-Hilfs-Tabellen.
27BD - 28B1 FN

28B2 - 2A51 Diese Routine sucht nach einer Variablen und läßt ihre Parameter auf dem Kalkulatorstapel zurück.
2A52 - 2AB0 Die Slicing-Routine.
2AB1 - 2AFE Die Inhalte von A, B, C, D und E werden auf den Kalkulatorstapel gebracht.
2AFF - 2BF0 LET
2BF1 - 2C01 Diese Routine holt den obersten Wert vom Kalkulator und lädt ihn in A, B, C, D und E.
2C02 - 2C87 DIM
2C88 - 2F9A Verschiedene Arithmetik-Routinen. Die wichtigsten sind
2D28 - 2D2A Der Wert in A wird auf den Kalkulatorstapel gebracht.
2D2B - 2D3A Der Wert von BC wird auf den Kalkulatorstapel gebracht.
2DA2 - 2DC0 Hier wird der letzte Wert vom Stapel geholt und in BC geladen.
2DD5 - 2DE2 Hier wird der letzte Wert des Kalkulatorstapels in A geladen.
2DE3 - 2F9A Diese Routine holt sich den letzten Wert vom Kalkulatorstapel und druckt sie im normalen Zahlenformat.

2F9B - 386D Der Kalkulator (Literale s. Anhang II).
2F9B - 300E Verschiedene Arithmetik-Routinen.
300F - 3013 Subtraktion. Der oberste Wert des Kalkulatorstapels wird vom zweiten abgezogen und das Ergebnis wird auf den Stapel gebracht.
3014 - 30A8 Addition. Die obersten Werte des Kalkulatorstapels werden addiert und abgelegt.
30CA - 31AE Multiplikation.
31AF - 3213 Division. Der oberste Wert wird durch den zweiten dividiert.
32C5 - 3206 Eine Tabelle, die folgende fünf Konstanten enthält: Null; Eins; 0,5; PI/2; Zehn. Der Aufruf dieser Literale bringt den entsprechenden Wert auf den Kalkulatorstapel.

32D7 - 335A Eine Adresstabelle des Kalkulators.
335B - 33A1 Dies ist die Kontrollroutine des Kalkulators.
33A2 - 36AE Verschiedene Kalkulator-Unterroutinen.
36AF - 386D Berechnung von SIN, ATN, EXP und LN.
386E - 3FFF Der Zeichensatz.

7. Benutzung des ROM

7.1 CLS

Um den Bildschirm zu löschen, können Sie die CLS-Routine mit CALL 0D6B aufrufen. Diese Routine führt die Funktion genau so aus wie das BASIC-CLS.

Es ist wichtig, vor Aufruf von CLS den Kanal auf 2 zu öffnen. Wenn weitere Ausgaben auf dem oberen Teil des Bildschirms getätigt werden sollen, muß der Kanal wieder auf 2 geöffnet werden. Um nun den Bildschirm zu löschen, benutzen Sie am besten folgende Routine:

```
ORG 30000
LD A,2
CALL 1601H      (Kanal öffnen)
CALL 0D6BH      (CLS)
RET

110 DATA "3E02CD0116CD6B0DC9"
```

Es besteht auch die Möglichkeit, nur einen Teil des Bildschirms zu löschen; dabei wird von der untersten Bildschirmzeile an gezählt, wieviele Zeilen gelöscht werden sollen; diese Anzahl muß in das B-Register geladen werden. Natürlich muß auch hier wieder der Kanal geöffnet werden. Um z.B. die untersten vier Zeilen zu löschen können Sie folgendes Programm verwenden:

```
ORG 30000
LD A,2
CAll 1601H      (Kanal öffnen)
LD B,4
CALL 0E44H      (Teil-CLS)
RET

110 DATA "3E02CD01160604CD440EC9"
```

7.2 Drucken

Als erstes müssen Sie hier bestimmen, ob Sie auf den oberen oder unteren Bildschirmteil drucken wollen, da Sie für den unteren den Kanal mit 1 und für den oberen den Kanal wie gewohnt mit 2 öffnen müssen.

Für den eigentlichen Zeichendruck brauchen Sie nur den Akkumulator mit dem Zeichencode laden und danach den Befehl RST 10 verwenden.

Im Akkumulator kann auch z.B. das AT-Steuerzeichen oder ein Farb-Steuerzeichen stehen. Danach benötigen Sie allerdings noch weitere RST 10 Befehle, da Sie der Unterroutine noch weitere Parameter wie Zeile, Spalte oder Farbe nachliefern müssen.

Hier ein paar Beispiele:

```
ORG 30000
LD A,2
CALL 1601H      (Kanal öffnen)
LD B,24         (Alle Zeilen)
CALL 0E44H      (Teil-CLS)
LD A,10H        (INK 2)
RST 10H
LD A,2
RST 10H
LD A,11H        (PAPER 6)
RST 10H
LD A,6
RST 10H
LD A,12H        (FLASH 1)
RST 10H
LD A,1
RST 10H
LD A,16H        (AT 11,3)
RST 10H
LD A,0BH
RST 10H
LD A,3
RST 10H
LD A,26H        (&)
RST 10H
RET
```

```
110 DATA "3E02CD01160618CD440E3E10D73E02D73E11D73E06D73E12D73E01D73E16D73E0BD73E03
D73E26D7C9"
```

Dieses Maschinencodeprogramm läßt sich durch folgende BASIC-Teile ersetzen:

```
CLS : PRINT INK 2; PAPER 6; FLASH 1; AT 11,3;"&"
```

7.2.1 Druck von Zeichenketten

Im Spectrum haben alle Strings folgende Parameter:

DE - Startadresse der Zeichenkette
BC - Länge der Zeichenkette

Laden Sie Ihre Stringparameter in die Doppelregister und benutzen Sie CAll 203C.

Das obere Beispiel können Sie mit dieser Routine kürzen: POKEn Sie die PRINT-Werte nacheinander ab z.B. Adresse 31000:

```
16 }
 2 } INK 2

17 }
 6 } PAPER 6

18 }
 1 } FLASH 1
```

```
22
11 } AT 11,3
 3
```

38 &

Dies sind genau 10 Werte. BC wird also mit 10 und DE mit 31000 geladen. Nun folgt das Programm zum Ausdruck auf dem Bildschirm:

```
ORG 30000
LD A,2
CALL 1601H      (Kanal öffnen)
LD DE,31000
LD BC,0AH
CALL 203CH      (String drucken)
RET

110 DATA "3E02CD0116111879010A00CD3C20C9"
```

7.2.2 Druck von Fließkommazahlen

Mit dieser Routine können Sie eine Zahl vom Kalkulatorstapel holen und sie ausdrucken. Selbstverständlich muß auch hier der Kanal geöffnet werden. Folgendes Programm druckt die Zahl PI:

```
ORG 30000
LD A,2
CALL 1601H      (Kanal öffnen)
RST 28H         (In den Kalkulator)
A3H             (PI/2 holen)
31H             (Duplizieren)
0FH             (Addieren)
38H             (Kalkulator verlassen)
CALL 2DE3H      (Drucke Zahl)
RET

110 DATA "3E02CD0116EFA3310F38CDE32DC9"
```

7.2.3 Druck von Integerzahlen

Diese Routine druckt Integer im Bereich von 0 - 9999. Hierzu brauchen Sie die Zahl, die Sie drucken möchten, nur in BC zu laden und CAll 1A1B benutzen. Um z.B. 1000 dez zu drucken, können Sie folgende Routine benutzen:

```
ORG 30000
LD A,2
CALL 1601H      (Kanal öffnen)
LD BC,1000
CALL 1A1BH      (Integer-Druck)
RET

11 DATA "3E02CD011601E803CD1B1AC9"
```

7.3 PLOT

Um einen Punkt auf den Bildschirm zu bringen, müssen Sie die x-Koordinate in das C-Register und die Y-Koordinate in das B-Register laden. Danach benutzen sie CALL 22E5. Das folgende Beispiel zeigt den Befehl PLOT 127,87:

```
ORG 30000
LD A,2
CALL 1601H      (Kanal öffnen)
LD B,87         (Y-Koordinate)
LD C,127        (X-Koordinate)
CALL 22E5H      (PLOT)
RET

110 DATA "3E02CD011606570E7FCDE522C9"
```

7.4 DRAW

Auch hier besteht die Möglichkeit, entweder gerade Linien oder Kreisbögen zu ziehen. Wenden wir uns zuerst den Linien zu:

Wenn Sie Linien entlang der X- oder Y-Achse ziehen, ist es besser, eine PLOT-Schleife zu programmieren, da diese schneller abgearbeitet wird.

Benutzen Sie CALL 24BA, wobei das B-Register ABS y, das C-Register ABS x, das D-Register SGN y und das E-Register SGN x enthalten muß.

Ganz wichtig! Bevor Sie DRAW benutzen, müssen Sie das H'L'-Registerpaar wegspeichern und bevor Sie ins BASIC zurückspringen, müssen Sie den Wert wieder in H'L' laden, da H'L' die BASIC-Rücksprungadresse enthält.

Das folgende Beispiel entspricht den BASIC-Befehlen

```
PLOT 0,175: DRAW 255,-175

ORG 30000
LD A,2
CALL 1601H
EXX
PUSH HL
EXX
LD B,175
LD C,0
CALL 22E5H      (PLOT)
LD B,175
LD C,255
LD D,255
LD E,1
CALL 24BAH      (DRAW)
EXX
POP HL
EXX
RET

110 DATA "3E02CD0116D9E5D906AF0E00CDE52206AF0EFF16FF1E01CDBA24D9E
1D9C9"
```

Um nun einen Kreisbogen zu ziehen, müssen Sie die drei Werte auf dem Kalkulatorstapel ablegen. Zuoberst muß das Bogenteil, darunter die Y- und dann die X-Koordinate stehen. Das heißt, daß Sie zuerst die X-, dann Y-Koordinate und dann den Bogen ablegen müssen. Danach verwenden Sie CALL 2394.

Dieses Beispiel entspricht PLOT 127,87: DRAW 20,20,PI

```
ORG 30000
LD A,2
CALL 1601H
EXX
PUSH HL
EXX
LD B,87
LD C,127
CALL 22E5H      (PLOT)
RST 28H         (In den Kalkulator)
A4H             (10 auf den Stapel)
31H             (Duplizieren)
0FH             (Addieren)
31H             (Duplizieren)
A3H             (PI/2 auf den Stapel)
31H             (Duplizieren)
0FH             (Addieren)
38H             (Kalkulator verlassen)
CALL 2394H      (DRAW)
EXX
POP HL
EXX
RET

110 DATA "3E02CD0116D9E5D906570E7FCDE522EFA4310F31A3310F38CD9423D
9E1D9C9"
```

7.5 CIRCLE

Auch hier müssen drei Parameter auf dem Kalkulatorstapel vorliegen, bevor CALL 232D benutzt wird. Die folgende Routine zeigt CIRCLE 40,40,10

```
ORG 30000
LD A,2
CALL 1601H      (Kanal öffnen)
EXX
PUSH HL
EXX
RST 28H         (In den Kalkulator)
A4H             (10 auf den Stapel)
31H             (Duplizieren)
0FH             (Addieren)
31H             (Duplizieren)
0FH             (Addieren)
31H             (Duplizieren)
A4H             (10 auf den Stapel)
38H             (Kalkulator verlassen)
CALL 232DH      (CIRCLE)
EXX
POP HL
EXX
RET

110 DATA "3E02CD0116D9E5D9EFA4310F310F31A438CD2D23D9E1D9C9"
```

7.6 Die beständigen Attribute

Wie schon in anderen Kapiteln erklärt, setzen sich die Attribute folgendermaßen zusammen:

Bit 0 - 2	INK
Bit 3 - 5	PAPER
Bit 6	BRIGHT
Bit 7	FLASH

Alle Bytes des Bild-Attributspeichers und auch die Systemvariablen BORDER, ATTR-T, MASK-P UND MASK-T haben diese Form.

Die Systemvariable P-Flag enthält die beständigen und zeitweiligen Flags für INK 9, PAPER 9, INVERSE 1 und OVER 1. Die geraden Bits (6,4,2,0) sind für die zeitweiligen und die ungeraden (7,5,3,1) für die beständigen oder konstanten Attribute.

7.6.1 BORDER

Die Bildschirm-BORDER-Farbe können Sie mit OUT 254,Farbe sowohl in BASIC als auch in Maschinencode ändern.

Um z.B. einen grünen Rand zu erhalten, verwenden Sie folgendes Programm:

```
ORG 30000
LD A,2
CALL 1601H      (Kanal öffnen)
LD A,4
OUT (254),A
RET

110 DATA "3E02CD01163E04D3FEC9"
```

7.6.2 PAPER

Der kürzeste Weg, um eine andere PAPER-Farbe zu bekommen, ist, die Bits 3 bis 5 durch SET und RES zu verändern.

Für PAPER 9 brauchen Sie nur Bit 7 von der Systemvariablen P-Flag setzen.

Bei PAPER 8 werden die Bits 3 - 5 von MASK-P geändert.

INK

Für INK werden die Bits 0 bis 2 durch SET und RES geändert.

Bei INK 9 wird Bit 5 von P-Flag gesetzt.

Bei INK 8 werden die Bits 0 - 2 von MASK-P geändert.

7.6.3 INVERSE

Um INVERSE 1 zu erreichen, müssen Sie Bit 3 von P-Flag setzen. Für INVERSE 0 brauchen Sie es nur zurückzusetzen.

7.6.4 OVER

Hier gilt das Gleiche für Bit 1 von P-Flag.

7.6.5 BRIGHT

Auch hier muß nur entsprechend Bit 6 von ATTR-P gesetzt werden. Für BRIGHT 8 muß Bit 6 von MASK-P gesetzt werden.

7.6.6 FLASH

Für diese Funktion muß Bit 7 von ATTR-P und für FLASH 8 Bit 7 von MASK-P geändert werden.

7.7 POINT

Auch hier werden die Koordinaten wie bei PLOT in B und C geladen und dann CALL 22CE benutzt.

Das nachfolgende Beispiel testet das Pixel 0,0 und gibt das Ergebnis nach BC. Natürlich können Sie auch die 'Stapel-in-A' Routine benutzen, aber Sie werden wahrscheinlich mit PRINT USR das Ergebnis ablesen wollen. Natürlich müssen Sie vor dem Aufruf der Routine einen Punkt plotten, da dies die Routine nicht erledigt.

```
ORG 30000
LD A,2
CALL 1601H
LD B,0
LD C,0
CALL 22CEH      (POINT (0,0))
CALL 2DA2H      (Ergebnis in BC)
RET

110 DATA "3E02CD011606000E00CDCE22CDA22DC9"
```

7.8 SCREEN$

Für diese Routine muß die Zeile nach C und die Spalte nach B geladen und darauf CALL 2538 benutzt werden. Danach finden Sie die Stringparameter auf dem Stapel. Nun wird die Parameter-Routine (2BF1) verwendet, die die Parameter vom Stapel holt. Da die Länge 1 ist, zeigt DE direkt auf das Zeichen. RST 10 dient zum Druck des Zeichens. Sollten Sie SCREEN$ in einem Maschinencodeprogramm öfter benutzen, kann es vorkommen, daß die Fehlermeldung 4 erscheint. Wenn dies der Fall ist, sollten Sie im Programm von Zeit zu Zeit die Stapel-Lösch-Routine bei 16C5 aufrufen.

```
ORG 30000
LD A,2
CALL 1601H
LD B,0
LD C,0
CALL 2538H      (SCREEN$ (0,0))
CALL 2BF1H      (Holt Parameter)
LD A,(DE)       (Zeichen in A)
RST 10H
RET

110 DATA "3E02CD011606000E00CD3825CDF12B1AD7C9"
```

7.9 ATTR

Auch hier werden Zeile und Spalte wieder nach BC geladen.

Wenn Sie sich POINT und SCREEN$ aufmerksam angesehen haben, gibt es hierzu nichts mehr zu sagen.

```
ORG 30000
LD A,2
CALL 1601H
LD B,0
LD C,0
CALL 2583H      (ATTR (0,0))
CALL 2DA2H      (Ergebnis in BC)
RET

110 DATA "3E02CD011606000E00CD8325CDA22DC9"
```

7.10 BEEP

Es gibt zwei Möglichkeiten, einen Ton zu erzeugen. Die erste, die wie im normalen BASIC arbeitet, verwendet den Stapel, so daß auch z.B. Brüche benutzt werden können. Wenn Sie, wie im Beispiel, erst 1 und dann 3 auf den Stapel bringen und 3F8 aufrufen, entspricht dies dem BASIC-Befehl BEEP 1,3.

Wenn Sie aber ein schnelles Maschinencodespiel schreiben möchten, ist diese Methode zu langsam, da sie den Stapel benutzt. Mit der zweiten Routine werden Sie ein wenig experimentieren müssen, da Sie in HL die Tonhöhe (die man ja nicht von jedem Ton weiß) und in DE die Dauer, die sich aus der Frequenz ergibt, laden.

Wie schon gesagt, gibt es hierfür kein 'Rezept'; man bekommt aber nach einigen Proben ein Gefühl dafür, wie die zwei Werte zusammenhängen. Das zweite Beispiel liefert einen kurzen hellen Ton.

```
ORG 30000
LD A,1          (1 Sekunde)
CALL 2D28H      (Auf den Stapel)
LD A,3          (Ton 3)
CALL 2D28H      (Auf den Stapel)
CALL 3F8H       (BEEP 1,3)
RET

110 DATA "3E01CD282D3E03CD282DCDF803C9"


ORG 30000
LD HL,FFH       (Länge)
LD DE,10H       (Tonhöhe)
CALL 3B5H       (BEEP)
RET

110 DATA "21FF00111000CDB503C9"
```

8. Beispielprogramme

8.1. Links-Scroll

Dieses Programm verschiebt mit LDIR jede der 192 hochauflösenden Zeilen um ein Zeichen nach links in der Schleife namens ALOOP.

In BLOOP wird die PRINT-Position 31 Zeile für Zeile mit einem Leerzeichen aufgefüllt.

Gehen Sie dieses Programm Zeile für Zeile durch und überlegen Sie, was bei den einzelnen MC-Befehlen geschieht.

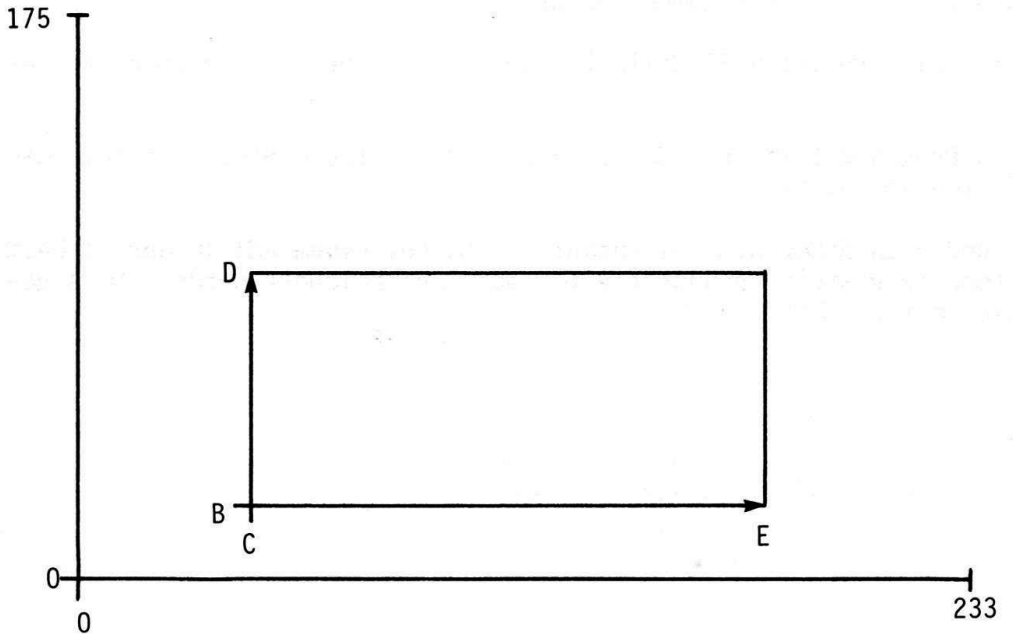
A dient in ALOOP und B in BLOOP als Zeilenzähler. Da bei einem RST 10 der Inhalt von den meisten Doppelregistern verändert wird, muß B gespeichert werden. Dies geschieht im Beispiel mit PUSH BC und POP BC.

```
ORG 30000
LD DE,4000H
LD HL,4001H
LD A,192
ALOOP LD BC,31
LDIR
INC DE
INC HL
DEC A
CP 0
JR NZ,ALOOP
LD A,2
CALL 1601H
LD B,0
BLOOP PUSH BC
LD A,16H
RST 10H
POP BC
PUSH BC
LD A,B
RST 10H
LD A,31
RST 10H
LD A,20H
RST 10H
POP BC
INC B
LD A,B
CP 22
JR NZ,BLOOP
RET
```

```
110 DATA "1100402101403EC0011F00EDB013233DFE0020F43E02CD01160600C53E16D7C1C578D73E
1FD73E20D7C10378FE1620EBC9"
```

8.2 Zeichnen und Ausfüllen eines hochauflösenden Rechtecks

Stellen Sie sich ein Rechteck auf dem Bildschirm vor, das Sie zeichnen und ausfüllen möchten. In der nachstehenden Grafik wird gezeigt, welche Register Parameter enthalten müssen:



Das Programm fängt in der linken unteren Ecke zu zeichnen an und zieht Zeile für Zeile nach oben durch. Im Programm wird PLOT-Routine des ROMs benutzt; die Verwendung der DRAW-Routine wäre natürlich einfacher, wobei aber man nicht die Schleifenstruktur des Programms sehen würde.

Zuerst wird die INK-Farbe festgelegt. Dann werden C und E gespeichert, da die beiden Werte in der Schleife auf Null gezählt werden, danach aber wieder benötigt werden. LOOP1 zählt hierbei die Zeilen des Rechtecks und LOOP2 die Punkte einer Zeile. Da bei PLOT-Routine des ROMs die Register verändert werden, müssen BC und DE zwischengespeichert werden, weil sie nach PLOT noch benötigt werden.

```
ORG 30000
LD A,2
CALL 1601H      (Kanal öffnen)
LD A,10H
RST 10H
LD A,3          (INK in A)
RST 10H
LD B,10         (Anfang Y)
LD C,30         (Anfang X)
LD D,110        (Höhe Y)
LD E,190        (Länge X)
LD HL,31001
LD (HL),C
INC HL
LD (HL),E
LOOP2 LD A,(31002)
LD E,A
LOOP1 PUSH DE
```

```
PUSH BC
CALL 22E5H      (PLOT)
POP BC
POP DE
INC C
DEC E
JR NZ,LOOP1
INC B
LD A, (31001)
LD C,A
DEC D
JR NZ,LOOP2
RET
```

```
110 DATA "3E02CD01163E10D73E03D7060A0E1E166E1EBE2119797123733A1A795FD5C5CDE522C1D1
0C1D20F5043A19794F1520E9C9"
```

9. Abschließend

Zum jetzigen Zeitpunkt werden Sie in der Lage sein, eigene Maschinencodeprogramme zu schreiben. Am Anfang werden Sie noch wahrscheinlich noch Schwierigkeiten teils der Logik und teils der Hilfsmittel wegen haben. Für das Schreiben von längeren Programmen sollten Sie sich unbedingt ein Assembler-Programm zulegen.

Nun noch ein paar Tips:

Am Anfang sollten Sie so oft wie möglich versuchen, die ROM-Routinen zu benutzen, da der Rechner dann meist nicht aussteigt, (d.h., daß er irgendwo eine Endlosschleife in Maschinencode abarbeitet, die man nicht unterbrechen kann,) sondern Fehlermeldungen ausdruckt. Diese Routinen sind zwar meist ein wenig langsamer als selbstgeschriebene, andererseits aber wesentlich sicherer.

Wenn Sie in einem Programm nach Fehlern suchen und prüfen wollen, ob der Rechner bis zu einer ganz bestimmten Stelle im Programm kommt, so setzen Sie an diese Stelle ein RST 8. Wenn der Rechner nun den Befehl RST 8 abarbeitet, verwendet er das darauffolgende Byte als Fehlercode und wirft eine Fehlermeldung aus (egal, wie komisch sie auch ist). Diese Methode ist absolut sicher; dies trifft unter anderem auch zu, wenn Sie sich in Unterroutinen etc. befinden. Für die Fehlersuche gibt es allerdings auch Hilfsprogramme, die meist mit einem Disassembler zusammen als Disassembler/Debug Programme (Debug=Entwanzer (orig. Übers.)) angeboten werden.

Im Übrigen wünsche ich Ihnen viel Erfolg beim Programmieren und vor allem viel Geduld bei der Fehlersuche.

Teil 2
Basic

1. BASIC-Befehle - Nützliches und Wissenswertes

ABS b — Nichts hinzuzufügen
ACS b — " "
AND b — " "
ASN b — " "
ATN b — " "

ATTR(a,b) — Diese Funktion ergibt den Attributwert des entsprechenden Zeichen, der folgendermaßen codiert wird:

```
INK+8*PAPER+64*BRIGHT+128*FLASH
```

BEEP a,b — Nichts hinzuzufügen
BIN b — " "

BORDER b — Man kann mit dem Befehl OUT 254,b, wobei b einen Wert 7 bis 0 annehmen kann, eine zeitweilige BORDER Änderung erreichen; sobald aber ein ENTER zum Bildschirmlöschen gedrückt wird, ändert sich der BORDER wieder in die Farbe, die er vorher hatte. Beispiel:

```
BORDER 0   ENTER   (schwarzer Border)
OUT 254,6  ENTER   (jetzt gelb)
ENTER              (wieder schwarz)
```

BORDER durch OUT bleibt nur bestehen, wenn man auch die Systemvariable BORDCR ändert.

BRIGHT b — Nichts hinzuzufügen

CAT — Da zu dem Zeitpunkt, zu dem ich dieses Buch schreibe, noch keine Microdrives erhältlich sind, läßt sich hierüber nichts sagen.

CHR$ b — Nichts hinzuzufügen
CIRCLE a,b,c — " "
CLEAR — " "

CLEAR b — Der kleinste Wert für b ist 23821; unter diesem Wert nimmt der Spectrum keine Eingabe mehr an. Der größte Wert für b ist 32767 bei 16K oder 65535 bei 48K. Setzen Sie z.B. einmal den RAMTOP mit CLEAR 32767 oder 65535 auf die höchste Speicherstelle und beobachten Sie, was mit den selbstdefinierbaren Zeichen geschieht. Schalten Sie in den G-Modus, geben Sie die Zeichen von A bis U ein und drücken dann wiederholt auf SPACE, wobei Sie auf die Buchstaben achten.

CLOSE — für Microdrives

CLS — setzt in alle Bytes von D-File (16384 bis 22527) Nullen und in alle Bytes des Attributspeichers (22528 bis 23295ü den Wert von ATTR-P. Dies können Sie überprüfen, indem Sie in ATTRP-P verschiedene Werte poken und danach wiederholt ENTER drücken.

CODE A$ — Nichts hinzuzufügen

COPY — Einer der wenigen Befehle, die die Realzeituhr während seiner Ausführung ausschaltet.

COS b — Nichts hinzuzufügen
DATA a$,b — " "

DEF FN a(b..c)=d
DEF FN a$(b.c)=d — Es ist interessant, zu wissen, daß folgende Programmzeilen bis jetzt der einzige Weg sind, den Spectrum im BASIC abstürzen zu lassen:

```
10 DEF FN a()=FN a()
20 PRINT FN a()
```

Führen Sie dieses Programm aus und warten Sie nur ein wenig.

DIM a(b,..,c) — Nichts hinzuzufügen
DRAW a,b — " "
DRAW a,b,c — " "
ERASE a$ — Für Microdrives

EXP b — Nichts hinzuzufügen

FLASH — "

FN a(b..c) — "

FOR a=b TO STEP d — Man sollte wissen, daß, wenn alle Werte, die b,c und d annehmen, INTEGER-Zahlen zwischen -65535 und 65535 sind, die Schleifensteuerung bis zu 20% schneller abläuft.

FORMAT a$ — Für Microdrives

GOSUB b — Nichts hinzuzufügen

GOTO b — "

IF a THEN q — "

IN b — Wenn der angesprochene Port nicht benutzt wird, ist der Wert, den Sie erhalten, 255, und nicht Null!

INK b — Nichts hinzuzufügen

INKEY$ — "

INPUT ... — Man kann alle PRINT-Steuerzeichen benutzen mit der Ausnahme von AT, da hierbei nur tabuliert wird. AT kann man aber trotzdem erreichen, wenn man den Ausdruck INPUT AT x,y in den Ausdruck INPUT AT x,0;AT 0,y auflöst.

INT b — Nichts hinzuzufügen

INVERSE b — "

LEN b — "

LET a=b — "

LIST b — Interessant hierbei ist, daß z.B. LIST 49151 das Gleiche wie LIST 1 bewirkt.

LLIST b — Nichts hinzuzufügen

LN b — "

LOAD a$ — Hierbei ist noch der Aufbau des Kopfes der beiden SAVE-Blöcke zu besprechen. Er ist 17 Bytes lang und besteht aus:

- Einem Informationsbyte, daß
  0 für BASIC
  1 für nummerisches Feld
  2 für alphanummerisches Feld
  3 für CODE
  enthält.
- Zehn-Byte-Name
- Zwei Bytes, die die Länge des zu SAVEnden Blocks enthalten.
- Zwei Bytes, die beim BASIC-Programm die Zahl von LINE und bei CODE die Startadresse enthalten.
- Zwei Bytes, die beim BASIC-Programm die Länge des Programms enthalten.

LPRINT ... — Nichts hinzuzufügen

MERGE a$ — "

MOVE a$,b$ — Für Microdrives

NEW — Dieser Befehl intialisiert alle Systemvariablen unterhalb von RAMTOP bis auf RAMTOP, P-RAMT, RASP, PIP UND UDG.

NEXT a — Nichts hinzuzufügen

NOT b — "

OPEN — Wird für Microdrives benötigt. Man kann OPEN und CLOSE aber auch bei dem normalen Spectrum benutzen. Versuchen Sie einmal die Eingabe von PRINT #6,"OK?". Es wird solange nichts gedruckt, bis Kanal 6 nicht geöffnet worden ist. Geben Sie OPEN #6,"S" und dann noch einmal den ersten Befehl ein. Nun kann der Befehl ausgeführt werden, bis Sie den Kanal mit CLOSE #6 wieder schliessen.
Der String bei OPEN kann folgendes bedeuten:

"K" Eingabe von Tastatur und Ausgabe auf dem unteren Teile des Bildschirms. Versuchen Sie

```
OPEN #1,"K":PRINT #1;AT 0,0;"NA?";
AT 3,5;INK 2;"GUT?"
```

"P" Ausgabe auf Drucker
OPEN #1,"P":PRINT #1;"Dies geht auch"
"S" Ausgabe auf dem oberen Teil des Bildschirms, also wie normales PRINT.

OR b	Nichts hinzuzufügen
OUT a,b	" "
OVER b	" "
PAPER b	" "
PAUSE b	" "
PEEK b	" "
PI	" "
PLOT	Mit OVER 1 kann UNPLOT simuliert werden; um keinen Punkt zu plotten, kann man INVERSE 1 benutzen. Als Attribut wird nur permanentes INK genommen und PAPER, FLASH und BRIGHT müssen im PLOT-Befehl stehen.
POINT (a,b)	Nichts hinzuzufügen
POKE a,b	" "
PRINT ...	" "
RANDOMIZE ..b	" "
READ ...	" "
REM	" "
RESTORE b	" "
RETURN	" "
RND	Die Zufallszahl wird folgendermaßen generiert: Als Startwert wird der letzte Wert aus SEED, der zwischen 0 und 65535 liegt, geholt. Hierbei werden alle Zahlen einmal durchgegangen, so daß es 65535 verschiedene Zufallszahlen gibt, worauf sich die Reihenfolge wiederholt. Nun wird SEED um eins erhöht und mit 75 multipliziert. Dann ist SEED gleich SEED modulo 65537. SEED wird um eins erniedrigt, RND ist SEED/65536.
RUN	Nichts hinzuzufügen
SAVE a$	" "
SCREEN$	Diese Funktion testet nur den normalen Zeichensatz und keine selbstdefinierten und keine Grafikzeichen, da diese nicht in 8x8-Bit-Form gespeichert sind. Dafür ist diese Funktion für den Spectrum sehr schnell.

Zu den Funktionen SGN b, SIN b, SQR b, STR$ b, STOP, TAN b, USR b und VERIFY a$ ist ebenfalls nichts hinzuzufügen.

2. Wie optimiere ich ein BASIC-Programm?

Dieses Kapitel ist in zwei Teile geteilt:

Der 1. Teil ist für diejenigen, die Speicherplatz sparen wollen es nicht relevant ist, daß das Programm eventuell etwas langsamer wird.
Der 2. Teil ist für diejenigen, denen es hauptsächlich auf die Geschwindigkeit des Programms ankommt.

1. Die wichtigsten Regeln beim Speicherplatzsparen sind folgende:

- möglichst viele Befehle in eine Zeile bringen, da dies Zeilennummern spart.
- möglichst kurze Variablennamen und möglichst wenig Variablen verwenden.
- Wenn jedoch eine Zahl mehr als 8 mal vorkommt, lohnt es sich, eine Variable zu benutzen, da jede "reelle" Zahl mindestens 7 Bytes benötigt.
- Sprünge wie GOTO 1000 etc. mit GOTO 1E3 abkürzen.
- Statt 0 kann man PI-PI (4Bytes weniger!) und statt 1 PI/PI schreiben.

2. Wenn eine hohe Programmgeschwindigkeit erwünscht ist, sollte man ein Programm unter folgenden Gesichtspunkten schreiben:

- Wenn eine Funktion nicht zu oft vorkommt, kann man sie besser ausschreiben anstatt DEF FN zu verwenden, da sie dann mindestens 3% schneller abläuft.
- DEF FN gehört immer an den Programmanfang, da jede Zeile, die zwischen Zeile 0 und der Definition liegt, den Aufruf der Definition um 2% langsamer macht.
- Felder sollte man immer zuerst definieren, da sie nicht hinten an die Variablen angehängt werden, wenn ein neuer Wert hinzukommt.
- Auch hier möglichst wenig Variablen benutzen, da der Rechner dann weniger suchen muß.
- Unterprogramme, die oft benutzt werden, an den Anfang setzen.

Ein schnelles Programm könnte dann folgendermaßen aussehen:

```
    1 GOTO 1E3
    2 DEF FN a() =..
    3 DEF FN b()=...
  usw.
  100 Ab hier Unterprogramme
  101
  102
  usw.
 1000 Ab hier das Hauptprogramm
 1001 GOSUB 9E3
 usw.
 9000 Datendefinition
 9001 DIM a(100,100)
 9199 RETURN
 9200 DATA ...
```

Für beide Fälle gilt:

Benutzen Sie möglichst logische Operatoren; um z.B. X auf Null zu testen geben Sie statt

```
IF X=0 THEN...
IF NOT X THEN...
```

ein oder statt

```
IF Z=1 THEN PRINT"???"
PRINT "???" AND Z=1
```

Teil 3
Anhang

Anhang I

A.1 Die Befehlsparameter und Befehlsadressen

Es gibt im Spectrum eine Befehlsparametertabelle von 1A7A - 1B15, die hier alphabetisch geordnet aufgefürht ist. Es ist dem Benutzer fast unmöglich, all diese Befehle in Maschinencode zu benutzen, aber Befehle ohne Parameter wie z.B. CLS können direkt angesprungen werden und arbeiten einwandfrei.

Befehl	Einsprungadresse	Befehlsparameter
BEEP	03F8	08,00
BORDER	2294	06,00
BRIGHT	nicht anspringbar	07
CAT	1793	00
CIRCLE	2320	09,05
CLEAR	1EAC	03
CLOSE	16E5	06,0D
CLS	0D68	00
CONTINUE	1E5F	00
COPY	0EAC	00
DATA	1E27	05
DEF FN	1F60	05
DIM	2002	05
DRAW	2382	09,05
ERASE	1793	0A,00
FLASH	nicht anspringbar	07
FOR	1D03	04;06;06,05
FORMAT	1793!	0A,00
GO SUB	1EED	06,00
GOTO	1E67	06,00
IF	1CF0	06;05
INK	nicht anspringbar	07
INPUT	2089	05
INVERSE	nicht anspringbar	07
LET	nicht anspringbar	01;02
LIST	17F9	05
LLIST	17F5	05
LOAD	nicht anspringbar	0B
LPRINT	1FC9	05
MERGE	nicht anspringbar	0B
MOVE	1793	0A;0A,00
NEW	11B7	00
NEXT	1DAB	04,00
OPEN	1736	06;0A,00
OUT	1E7A	08,00
OVER	nicht anspringbar	07
PAPER	nicht anspringbar	07
PAUSE	1F3A	06,00
PLOT	22DC	09,00
POKE	1E80	08,00
PRINT	1FCD	05
RANDOMIZE	1E4F	03
READ	1DED	05
REM	1BB2	05
RESTORE	1E42	03
RETURN	1F23	00
RUN	1EA1	03
SAVE	nicht anspringbar	0B

STOP	1CEE	00
VERIFY	nicht anspringbar	0B

Das Komma bei den Befehlsparametern unterteilt mögliche Versionen und das Semikolon reiht notwendige Parameter auf.

Die Befehlsparameter bedeuten:

00 - Keine weiteren Operanden notwendig.
01 - Eine Variable wird verlangt.
02 - Ein numerischer oder alphanumerischer Ausdruck wird verlangt.
03 - Ein nummerischer Ausdruck kann hier stehen. Falls nicht, wird Null angenommen.
04 - Eine Variable mit einem Buchstaben wird verlangt.
05 - Eine beliebige Zeichenkette kann hier stehen.
06 - Ein numerischer Ausdruck wird verlangt.
07 - Farbcode.
08 - Zwei durch ein Komma getrennte, numerische Ausdrücke müssen hier stehen.
09 - wie 08, doch davor können Farbcodes stehen.
0A - Ein alphanumerischer Ausdruck wird verlangt.
0B - Für Kassettenbefehle.

Anhang II

A.2 Die Literale

Um den Kalkulator zu nutzen, müssen Sie ihn mit RST 28 aufrufen. Die nachfolgenden Werte sind dann Datenbytes, entweder Literale oder eine Fließkommazahl in ihrer Kurzform.

Die wichtigsten Literale sind:

01 - Austauschen der obersten Stapelwerte
02 - Löschen des obersten Stapelwertes
03 - Subtraktion
04 - Multiplikation
05 - Division
06 - Potenzieren
07 - OR
0F - Addition
1B - Negieren
1C - CODE
1D - VAL
1E - LEN
1F - SIN
20 - COS
21 - TAN
22 - ASN
23 - ACS
24 - ATN
25 - LN
26 - EXP
27 - INT
28 - SQR
29 - SGN
2A - ABS
2B - PEEK
2C - IN
2D - USR
30 - NOT
31 - Duplizieren des obersten Wertes
32 - Modulo-Division
34 - Nachfolgende Werte als Zahl auf den Stapel
38 - Verlassen des Kalkulators

Nun zur Darstellung einer Fließkommazahl in ihrer Kurzform:

Wenn man eine Fließkommazahl in ihrer Kurzform hat, kann man sie mit Hilfe des Literals 34 in ihre normale Form auf den Stapel bringen. Man kann auch, wenn die Bytes irgendwo im Stapel abgespeichert sind, DE mit der Zieladresse und H'L' mit der Startadresse der Kurzform in die Normalform bringen; dazu ruft man dann 33C6 auf.

Nun zur Kurzform:

Das erste Byte der Kurzform wird durch 64 dez dividiert und der Exponent ist

- wenn ein Rest übrigbleibt: Der Rest + 80 dez
- wenn kein Rest übrigbleibt: Das zweite Byte + 80 dez

Der Quotient, der immer 0,1,2 oder 3 ist, zeigt, wieviele weitere Bytes von der Kurzform genommen werden. Hierbei muß man natürlich den Quotienten um eins erhöhen. Alle fehlenden Bytes werden durch 0 ersetzt.

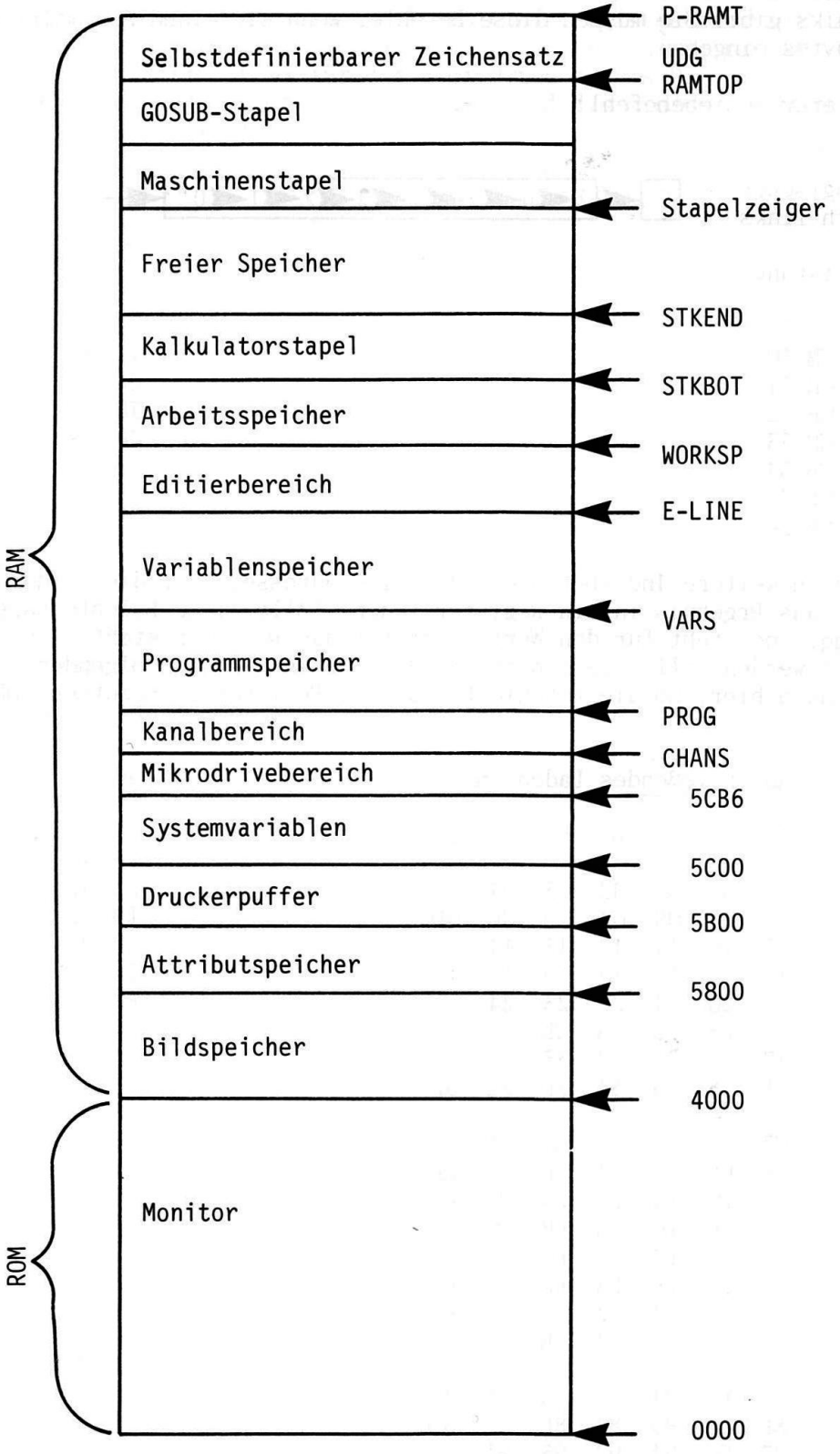
Hier zwei Beispiele:

0 : Kurzform 00 192 00
00 wird durch 64 dividiert. Es bleibt kein Rest, also ist der Exponent 192+64=256=0. Die Mantisse ist das dritte Byte, also auch 0. Der Rest wird aufgefüllt: 0 0 0 0 0

1 : Kurzform 64 192 0 1
Wie oben ist der Exponent 0. Der Quotient ist 1, das heißt, dass die nächsten zwei Bytes Mantisse sind. Heraus kommt 0 0 1 0 0.
Dies ist die Integer- und nicht die Fließkommaform.

Anhang III

A.3 Der Speicher



Anhang IV

A.4 Weitere Z80-Befehle

Die unten aufgeführten Z80-Befehle sind kein Standart, weshalb sie auch in keinem Assemblerhandbuch zu finden sind; kein Assembler kennt sie, da es auch keine Standart-Mnemoniks gibt. Sie müssen diese Befehle, wenn sie einen Assembler benutzen, als Datenbytes eingeben.

Zuerst ein weiterer Schiebebefehl:

SLIA
Invertiertes logisches
Verschieben nach links

Die Befehlsauflistung:

SLIA A	CB 37
SLIA B	CB 30
SLIA C	CB 31
SLIA D	CB 32
SLIA E	CB 33
SLIA H	CB 34
SLIA L	CB 35
SLIA (HL)	CB 36

Dann gibt es noch weitere Indexbefehle, die eine Adresse, auf die IX+dd zeigt, verändern und das Ergebnis in ein Register laden. Alle diese Befehle haben die Form DD CB dd qq. qq steht für den Wert. der für das Register steht, in das das Ergebnis geladen werden soll. Diese Werte finden Sie in der nachfolgenden Tabelle. Natürlich kann auch hier, um die Befehle für das IY-Register zu benutzen. DD in FD geändert werden.

	anschließendes Laden in						
Befehl	A	B	C	D	E	H	L
RLC (IX+dd)	07	00	01	02	03	04	05
RRC (IX+dd)	0F	08	09	0A	0B	0C	0D
RL (IX+dd)	17	10	11	12	13	14	15
RR (IX+dd)	1F	18	19	1A	1B	1C	1D
SLA (IX+dd)	27	20	21	22	23	24	25
SRA (IX+dd)	2F	28	29	2A	2B	2C	2D
SLIA (IX+dd)	37	30	31	32	33	34	35
SRL (IX+dd)	3F	38	39	3A	3B	3C	3D
SET 0,(IX+dd)	C7	C0	C1	C2	C3	C4	C5
SET 1,(IX+dd)	CF	C8	C9	CA	CB	CC	CD
SET 2,(IX+dd)	D7	D0	D1	D2	D3	D4	D5
SET 3,(IX+dd)	DF	D8	D9	DA	DB	DC	DD
SET 4,(IX+dd)	E7	E0	E1	E2	E3	E4	E5
SET 5,(IX+dd)	EF	E8	E9	EA	EB	EC	ED
SET 6,(IX+dd)	F7	F0	F1	F2	F3	F4	F5
SET 7,(IX+dd)	FF	F8	F9	FA	FB	FC	FD
RES 0,(IX+dd)	87	80	81	82	83	84	85
RES 1,(IX+dd)	8F	88	89	8A	8B	8C	8D
RES 2,(IX+dd)	97	90	91	92	93	94	95
RES 3,(IX+dd)	9F	98	99	9A	9B	9C	9D

RES 4,(IX+dd)	A7	A0	A1	A2	A3	A4	A5
RES 5,(IX+dd)	AF	A8	A9	AA	AB	AC	AD
RES 6,(IX+dd)	B7	B0	B1	B2	B3	B4	B5
RES 7,(IX+dd)	BF	B8	B9	BA	BB	BC	BD

Die interessanteste Gruppe von zusätzliches Befehlen ist folgende: Mit diesen Befehlen können Sie die Indexregister als Einzelregister benutzen. Auch hier wurde sich nur für die Darstellung des IX-Registers entschieden; für das IY-Register ist wieder FD zu benutzen. Da es für die zwei Bytes des IX-Registers keine Namen gibt, wurde das höherwertige Byte von mir mit Z und das niederwertige Byte mit X bezeichnet. Die möglichen Befehle sind dann:

LD X,dd	DD 2E dd
LD X,A	DD 6F
LD X,B	DD 68
LD X,D	DD 6A
LD X,E	DD 6B
LD X,Z	DD 6C
LD A,X	DD 7D
LD B,X	DD 45
LD C,X	DD 4D
LD D,X	DD 55
LD E,X	DD,5D
INC X	DD 2C
DEC X	DD 2D
ADD A,X	DD 85
ADC A,X	DD 8D
SUB X	DD 95
SBC	DD 9D
AND X	DD A5
XOR X	DD AD
OR X	DD B5
CP X	DD BD
LD Z,dd	DD 26 dd
LD Z,A	DD 67
LD Z,B	DD 60
LD Z,C	DD 61
LD Z,D	DD 62
LD Z,E	DD 63
LD Z,X	DD 65
LD A,Z	DD 7C
LD B,Z	DD 44
LD C,Z	DD 4C
LD D,Z	DD 54
LD E,Z	DD 5C
INC Z	DD 24
DEC Z	DD 25
ADD A,Z	DD 84
ADC A,Z	DD 8C
SUB Z	DD 94
SBC Z	DD 9C
AND Z	DD A4
XOR Z	DD AC
OR Z	DD B4
CP Z	DD BC

Anhang V

A.5 Fehler im ROM

Da das ROM ca. 7000 Maschinencodebefehle enthält, ist es kein Wunder, wenn einlige Fehler beim Schreiben hineingeraten sind. Diese Fehler werden meistens erst per Zufall nach einiger Zeit entdeckt. Hier sind die bis jetzt bekannten aufgelistet:

-Es ist nicht immer möglich, den Kontrollcode für CURSOR LINKS, also CHR$ 8 zu benutzen. Dies funktioniert nur, wenn Sie sich auf Zeile 1 bis 21 befinden.

-Den Kontrollcode für CURSOR RECHTS, also CHR$ 9 ist überhaupt nicht zu verwenden, da vergessen wurde, die neue PRINT-Position abzuspeichern.
-Die Verkettung von der Funktion SCREEN$ kann zu bösen Überraschungen führen.

Versuchen Sie:

```
PRINT AT 0,0;"ABCDEF"    ENTER
PRINT SCREEN$(0,0)+SCREEN$(0,1)    ENTER
```

Sie erwarten, daß hierbei AB herauskommt. Dies sollte auch der Fall sein; es wird aber BB gedruckt. Um dies zu umgehen, müssen Sie den SCREEN$s-Ergebnissen verschiedene Variablen zuweisen, die Sie dann verketten können.
-Auch bei STR$ im Bereich von -1 und 1 kann es Probleme geben.

Versuchen Sie folgende Eingabe:

```
PRINT "10"+STR$ .10
```

Auch dies kann man durch Variablen umgehen.
-Folgender Fehler ist auch sehr komisch: Geben Sie

```
1 PRINT 0: GO TO 1
```

gefolgt von RUN ein. Wenn das Programm "scroll?" fragt, drücken sie CAPS SHIFT und SYMBOL SHIFT. Es erscheint unten der Befehl RUN mit dem Kursor. Drücken Sie nun ENTER...
-Es ist auch möglich, eine Zeile samt Zeilenzeiger zu editieren.
Wenn Sie obige Zeile 1 eingegeben haben, geben Sie 2 ENTER ein.
Drücken sie nun auf EDIT.
-Sie sollten auch darauf achten, niemals irgendetwas zu CLOSEn, bevor Sie es nicht geOPENnd haben. Versuchen Sie einmal CLOSE #10. Es erscheint entweder eine unnormale Fehlermeldung oder das gesamte System stürzt ab.

* UMWANDLUNGSTABELLE *

* DEZ./HEX./BIN./2ER-KOMPL. *

0	00	00000000	64	40	01000000
1	01	00000001	65	41	01000001
2	02	00000010	66	42	01000010
3	03	00000011	67	43	01000011
4	04	00000100	68	44	01000100
5	05	00000101	69	45	01000101
6	06	00000110	70	46	01000110
7	07	00000111	71	47	01000111
8	08	00001000	72	48	01001000
9	09	00001001	73	49	01001001
10	0A	00001010	74	4A	01001010
11	0B	00001011	75	4B	01001011
12	0C	00001100	76	4C	01001100
13	0D	00001101	77	4D	01001101
14	0E	00001110	78	4E	01001110
15	0F	00001111	79	4F	01001111
16	10	00010000	80	50	01010000
17	11	00010001	81	51	01010001
18	12	00010010	82	52	01010010
19	13	00010011	83	53	01010011
20	14	00010100	84	54	01010100
21	15	00010101	85	55	01010101
22	16	00010110	86	56	01010110
23	17	00010111	87	57	01010111
24	18	00011000	88	58	01011000
25	19	00011001	89	59	01011001
26	1A	00011010	90	5A	01011010
27	1B	00011011	91	5B	01011011
28	1C	00011100	92	5C	01011100
29	1D	00011101	93	5D	01011101
30	1E	00011110	94	5E	01011110
31	1F	00011111	95	5F	01011111
32	20	00100000	96	60	01100000
33	21	00100001	97	61	01100001
34	22	00100010	98	62	01100010
35	23	00100011	99	63	01100011
36	24	00100100	100	64	01100100
37	25	00100101	101	65	01100101
38	26	00100110	102	66	01100110
39	27	00100111	103	67	01100111
40	28	00101000	104	68	01101000
41	29	00101001	105	69	01101001
42	2A	00101010	106	6A	01101010
43	2B	00101011	107	6B	01101011
44	2C	00101100	108	6C	01101100
45	2D	00101101	109	6D	01101101
46	2E	00101110	110	6E	01101110
47	2F	00101111	111	6F	01101111
48	30	00110000	112	70	01110000
49	31	00110001	113	71	01110001
50	32	00110010	114	72	01110010
51	33	00110011	115	73	01110011
52	34	00110100	116	74	01110100
53	35	00110101	117	75	01110101
54	36	00110110	118	76	01110110
55	37	00110111	119	77	01110111
56	38	00111000	120	78	01111000
57	39	00111001	121	79	01111001
58	3A	00111010	122	7A	01111010
59	3B	00111011	123	7B	01111011
60	3C	00111100	124	7C	01111100
61	3D	00111101	125	7D	01111101
62	3E	00111110	126	7E	01111110
63	3F	00111111	127	7F	01111111

* UMWANDLUNGSTABELLE HIGHER-BYTE *

00.....0	40...16384	80...32768	C0...49152
01.....256	41...16640	81...33024	C1...49408
02.....512	42...16896	82...33280	C2...49664
03.....768	43...17152	83...33536	C3...49920
04....1024	44...17408	84...33792	C4...50176
05....1280	45...17664	85...34048	C5...50432
06....1536	46...17920	86...34304	C6...50688
07....1792	47...18176	87...34560	C7...50944
08....2048	48...18432	88...34816	C8...51200
09....2304	49...18688	89...35072	C9...51456
0A....2560	4A...18944	8A...35328	CA...51712
0B....2816	4B...19200	8B...35584	CB...51968
0C....3072	4C...19456	8C...35840	CC...52224
0D....3328	4D...19712	8D...36096	CD...52480
0E....3584	4E...19968	8E...36352	CE...52736
0F....3840	4F...20224	8F...36608	CF...52992
10....4096	50...20480	90...36864	D0...53248
11....4352	51...20736	91...37120	D1...53504
12....4608	52...20992	92...37376	D2...53760
13....4864	53...21248	93...37632	D3...54016
14....5120	54...21504	94...37888	D4...54272
15....5376	55...21760	95...38144	D5...54528
16....5632	56...22016	96...38400	D6...54784
17....5888	57...22272	97...38656	D7...55040
18....6144	58...22528	98...38912	D8...55296
19....6400	59...22784	99...39168	D9...55552
1A....6656	5A...23040	9A...39424	DA...55808
1B....6912	5B...23296	9B...39680	DB...56064
1C....7168	5C...23552	9C...39936	DC...56320
1D....7424	5D...23808	9D...40192	DD...56576
1E....7680	5E...24064	9E...40448	DE...56832
1F....7936	5F...24320	9F...40704	DF...57088
20....8192	60...24576	A0...40960	E0...57344
21....8448	61...24832	A1...41216	E1...57600
22....8704	62...25088	A2...41472	E2...57856
23....8960	63...25344	A3...41728	E3...58112
24....9216	64...25600	A4...41984	E4...58368
25....9472	65...25856	A5...42240	E5...58624
26....9728	66...26112	A6...42496	E6...58880
27....9984	67...26368	A7...42752	E7...59136
28...10240	68...26624	A8...43008	E8...59392
29...10496	69...26880	A9...43264	E9...59648
2A...10752	6A...27136	AA...43520	EA...59904
2B...11008	6B...27392	AB...43776	EB...60160
2C...11264	6C...27648	AC...44032	EC...60416
2D...11520	6D...27904	AD...44288	ED...60672
2E...11776	6E...28160	AE...44544	EE...60928
2F...12032	6F...28416	AF...44800	EF...61184
30...12288	70...28672	B0...45056	F0...61440
31...12544	71...28928	B1...45312	F1...61696
32...12800	72...29184	B2...45568	F2...61952
33...13056	73...29440	B3...45824	F3...62208
34...13312	74...29696	B4...46080	F4...62464
35...13568	75...29952	B5...46336	F5...62720
36...13824	76...30208	B6...46592	F6...62976
37...14080	77...30464	B7...46848	F7...63232
38...14336	78...30720	B8...47104	F8...63488
39...14592	79...30976	B9...47360	F9...63744
3A...14848	7A...31232	BA...47616	FA...64000
3B...15104	7B...31488	BB...47872	FB...64256
3C...15360	7C...31744	BC...48128	FC...64512
3D...15616	7D...32000	BD...48384	FD...64768
3E...15872	7E...32256	BE...48640	FE...65024
3F...16128	7F...32512	BF...48896	FF...65280

128	80	10000000	-128	192	C0	11000000	-64
129	81	10000001	-127	193	C1	11000001	-63
130	82	10000010	-126	194	C2	11000010	-62
131	83	10000011	-125	195	C3	11000011	-61
132	84	10000100	-124	196	C4	11000100	-60
133	85	10000101	-123	197	C5	11000101	-59
134	86	10000110	-122	198	C6	11000110	-58
135	87	10000111	-121	199	C7	11000111	-57
136	88	10001000	-120	200	C8	11001000	-56
137	89	10001001	-119	201	C9	11001001	-55
138	8A	10001010	-118	202	CA	11001010	-54
139	8B	10001011	-117	203	CB	11001011	-53
140	8C	10001100	-116	204	CC	11001100	-52
141	8D	10001101	-115	205	CD	11001101	-51
142	8E	10001110	-114	206	CE	11001110	-50
143	8F	10001111	-113	207	CF	11001111	-49
144	90	10010000	-112	208	D0	11010000	-48
145	91	10010001	-111	209	D1	11010001	-47
146	92	10010010	-110	210	D2	11010010	-46
147	93	10010011	-109	211	D3	11010011	-45
148	94	10010100	-108	212	D4	11010100	-44
149	95	10010101	-107	213	D5	11010101	-43
150	96	10010110	-106	214	D6	11010110	-42
151	97	10010111	-105	215	D7	11010111	-41
152	98	10011000	-104	216	D8	11011000	-40
153	99	10011001	-103	217	D9	11011001	-39
154	9A	10011010	-102	218	DA	11011010	-38
155	9B	10011011	-101	219	DB	11011011	-37
156	9C	10011100	-100	220	DC	11011100	-36
157	9D	10011101	-99	221	DD	11011101	-35
158	9E	10011110	-98	222	DE	11011110	-34
159	9F	10011111	-97	223	DF	11011111	-33
160	A0	10100000	-96	224	E0	11100000	-32
161	A1	10100001	-95	225	E1	11100001	-31
162	A2	10100010	-94	226	E2	11100010	-30
163	A3	10100011	-93	227	E3	11100011	-29
164	A4	10100100	-92	228	E4	11100100	-28
165	A5	10100101	-91	229	E5	11100101	-27
166	A6	10100110	-90	230	E6	11100110	-26
167	A7	10100111	-89	231	E7	11100111	-25
168	A8	10101000	-88	232	E8	11101000	-24
169	A9	10101001	-87	233	E9	11101001	-23
170	AA	10101010	-86	234	EA	11101010	-22
171	AB	10101011	-85	235	EB	11101011	-21
172	AC	10101100	-84	236	EC	11101100	-20
173	AD	10101101	-83	237	ED	11101101	-19
174	AE	10101110	-82	238	EE	11101110	-18
175	AF	10101111	-81	239	EF	11101111	-17
176	B0	10110000	-80	240	F0	11110000	-16
177	B1	10110001	-79	241	F1	11110001	-15
178	B2	10110010	-78	242	F2	11110010	-14
179	B3	10110011	-77	243	F3	11110011	-13
180	B4	10110100	-76	244	F4	11110100	-12
181	B5	10110101	-75	245	F5	11110101	-11
182	B6	10110110	-74	246	F6	11110110	-10
183	B7	10110111	-73	247	F7	11110111	-9
184	B8	10111000	-72	248	F8	11111000	-8
185	B9	10111001	-71	249	F9	11111001	-7
186	BA	10111010	-70	250	FA	11111010	-6
187	BB	10111011	-69	251	FB	11111011	-5
188	BC	10111100	-68	252	FC	11111100	-4
189	BD	10111101	-67	253	FD	11111101	-3
190	BE	10111110	-66	254	FE	11111110	-2
191	BF	10111111	-65	255	FF	11111111	-1

* Z80 - BEFEHLSSATZ *

		nach CB hex	nach ED hex
0	NOP	RLC B	
1	LD BC,dddd	RLC C	
2	LD (BD),A	RLC D	
3	INC BC	RLC E	
4	INC B	RLC H	
5	DEC B	RLC L	
6	LD B,dd	RLC (HL)	
7	RLCA	RLC A	
8	EX AF,A'F'	RRC B	
9	ADD HL,BC	RRC C	
10	LD A,(BC)	RRC D	
11	DEC BC	RRC E	
12	INC C	RRC H	
13	DEC C	RRC L	
14	LD C,dd	RRC (HL)	
15	RRCA	RRC A	
16	DJNZ e	RL B	
17	LD DE,dddd	RL C	
18	LD (DE),A	RL D	
19	INC DE	RL E	
20	INC D	RL H	
21	DEC D	RL L	
22	LD D,dd	RL (HL)	
23	RLA	RL A	
24	JR e	RR B	
25	ADD HL,DE	RR C	
26	LD A,(DE)	RR D	
27	DEC DE	RR E	
28	INC E	RR H	
29	DEC E	RR L	
30	LD E,dd	RR (HL)	
31	RRA	RR A	
32	JR NZ,e	SLA B	
33	LD HL,dddd	SLA C	
34	LD (dddd),HL	SLA D	
35	INC HL	SLA E	
36	INC H	SLA H	
37	DEC H	SLA L	
38	LD H,dd	SLA (HL)	
39	DAA	SLA A	
40	JR Z,e	SRA B	
41	ADD HL,HL	SRA C	
42	LD HL,(dddd)	SRA D	
43	DEC HL	SRA E	
44	INC L	SRA H	
45	DEC L	SRA L	
46	LD L,dd	SRA (HL)	
47	CPL	SRA A	
48	JR NC,e		
49	LD SP,dddd		
50	LD(dddd),A		
51	INC SP		
52	INC (HL)		
53	DEC (HL)		
54	LD (HL),dd		
55	SCF		
56	JR C,e	SRL B	
57	ADD HL,SP	SRL C	
58	LD A,(dddd)	SRL D	
59	DEC SP	SRL E	
60	INC A	SRL H	
61	DEC A	SRL L	
62	LD A,dd	SRL (HL)	
63	CCF	SRL A	

		nach CB hex	nach ED hex
64	LD B,B	BIT 0,B	IN B,(C)
65	LD B,C	BIT 0,C	OUT (C),B
66	LD B,D	BIT 0,D	SBC HL,BC
67	LD B,E	BIT 0,E	LD (dddd),BC
68	LD B,H	BIT 0,H	NEG
69	LD B,L	BIT 0,L	RETN
70	LD B,(HL)	BIT 0,(HL)	IM 0
71	LD B,A	BIT 0,A	LD I,A
72	LD C,B	BIT 1,B	IN C,(C)
73	LD C,C	BIT 1,C	OUT (C),C
74	LD C,D	BIT 1,D	ADC HL,BC
75	LD C,E	BIT 1,E	LD BC,(dddd)
76	LD C,H	BIT 1,H	RETI
77	LD C,L	BIT 1,L	
78	LD C,(HL)	BIT 1,(HL)	
79	LD C,A	BIT 1,A	LD R,A
80	LD D,B	BIT 2,B	IN D,(C)
81	LD D,C	BIT 2,C	OUT (C),D
82	LD D,D	BIT 2,D	SBC HL,DE
83	LD D,E	BIT 2,E	LD (dddd),DE
84	LD D,H	BIT 2,H	
85	LD D,L	BIT 2,L	
86	LD D,(HL)	BIT 2,(HL)	IM 1
87	LD D,A	BIT 2,A	LD A,I
88	LD E,B	BIT 3,B	IN E,(C)
89	LD E,C	BIT 3,C	OUT (C),E
90	LD E,D	BIT 3,D	ADC HL,DE
91	LD E,E	BIT 3,E	LD DE,(dddd)
92	LD E,H	BIT 3,H	
93	LD E,L	BIT 3,L	
94	LD E,(HL)	BIT 3,(HL)	IM 2
95	LD E,A	BIT 3,A	LD A,R
96	LD H,B	BIT 4,B	IN H,(C)
97	LD H,C	BIT 4,C	OUT (C),H
98	LD H,D	BIT 4,D	SBC HL,HL
99	LD H,E	BIT 4,E	LD (dddd),HL
100	LD H,H	BIT 4,H	
101	LD H,L	BIT 4,L	
102	LD H,(HL)	BIT 4,(HL)	
103	LD H,A	BIT 4,A	RRD
104	LD L,B	BIT 5,B	IN L,(C)
105	LD L,C	BIT 5,C	OUT (C),L
106	LD L,D	BIT 5,D	ADC HL,HL
107	LD L,E	BIT 5,E	LD DE,(dddd)
108	LD L,H	BIT 5,H	
109	LD L,L	BIT 5,L	
110	LD L,(HL)	BIT 5,(HL)	
111	LD L,A	BIT 5,A	RLD
112	LD (HL),B	BIT 6,B	
113	LD (HL),C	BIT 6,C	
114	LD (HL),D	BIT 6,D	SBC HL,SP
115	LD (HL),E	BIT 6,E	LD (dddd),SP
116	LD (HL),H	BIT 6,H	
117	LD (HL),L	BIT 6,L	
118	HALT	BIT 6,(HL)	
119	LD (HL),A	BIT 6,A	
120	LD A,B	BIT 7,B	IN A,(C)
121	LD A,C	BIT 7,C	OUT (C),A
122	LD A,D	BIT 7,D	ADC HL,SP
123	LD A,E	BIT 7,E	LD SP,(dddd)
124	LD A,H	BIT 7,H	
125	LD A,L	BIT 7,L	
126	LD A,(HL)	BIT 7,(HL)	
127	LD A,A	BIT 7,A	

		nach CB hex	nach ED hex
128	ADD A,B	RES 0,B	
129	ADD A,C	RES 0,C	
130	ADD A,D	RES 0,D	
131	ADD A,E	RES 0,E	
132	ADD A,H	RES 0,H	
133	ADD A,L	RES 0,L	
134	ADD A,(HL)	RES 0,(HL)	
135	ADD A,A	RES 0,A	
136	ADC A,B	RES 1,B	
137	ADC A,C	RES 1,C	
138	ADC A,D	RES 1,D	
139	ADC A,E	RES 1,E	
140	ADC A,H	RES 1,H	
141	ADC A,L	RES 1,L	
142	ADC A,(HL)	RES 1,(HL)	
143	ADC A,A	RES 1,A	
144	SUB B	RES 2,B	
145	SUB C	RES 2,C	
146	SUB D	RES 2,D	
147	SUB E	RES 2,E	
148	SUB H	RES 2,H	
149	SUB L	RES 2,L	
150	SUB (HL)	RES 2,(HL)	
151	SUB A	RES 2,A	
152	SBC A,B	RES 3,B	
153	SBC A,C	RES 3,C	
154	SBC A,D	RES 3,D	
155	SBC A,E	RES 3,E	
156	SBC A,H	RES 3,H	
157	SBC A,L	RES 3,L	
158	SBC A,(HL)	RES 3,(HL)	
159	SBC A,A	RES 3,A	
160	AND B	RES 4,B	LDI
161	AND C	RES 4,C	CPI
162	AND D	RES 4,D	INI
163	AND E	RES 4,E	OUTI
164	AND H	RES 4,H	
165	AND L	RES 4,L	
166	AND (HL)	RES 4,(HL)	
167	AND A	RES 4,A	
168	XOR B	RES 5,B	LDD
169	XOR C	RES 5,C	CPD
170	XOR D	RES 5,D	IND
171	XOR E	RES 5,E	OUTD
172	XOR H	RES 5,H	
173	XOR L	RES 5,L	
174	XOR (HL)	RES 5,(HL)	
175	XOR A	RES 5,A	
176	OR B	RES 6,B	LDIR
177	OR C	RES 6,C	CPIR
178	OR D	RES 6,D	INIR
179	OR E	RES 6,E	OTIR
180	OR H	RES 6,H	
181	OR L	RES 6,L	
182	OR (HL)	RES 6,(HL)	
183	OR A	RES 6,A	
184	CP B	RES 7,B	LDDR
185	CP C	RES 7,C	CPDR
186	CP D	RES 7,D	INDR
187	CP E	RES 7,E	OTDR
188	CP H	RES 7,H	
189	CP L	RES 7,L	
190	CP (HL)	RES 7,(HL)	
191	CP A	RES 7,A	

		nach CB hex	nach ED hex
192	RET NZ	SET 0,B	
193	POP BC	SET 0,C	
194	JP NZ,dddd	SET 0,D	
195	JP dddd	SET 0,E	
196	CALL NZ,dddd	SET 0,H	
197	PUSH BC	SET 0,L	
198	ADD A,dd	SET 0,(HL)	
199	RST 0	SET 0,A	
200	RET Z	SET 1,B	
201	RET	SET 1,C	
202	JP Z,dddd	SET 1,D	
203		SET 1,E	
204	CALL Z,dddd	SET 1,H	
205	CALL dddd	SET 1,L	
206	ADC A,dd	SET 1,(HL)	
207	RST 8	SET 1,A	
208	RET NC	SET 2,B	
209	POP DE	SET 2,C	
210	JP NC,dddd	SET 2,D	
211	OUT dd,A	SET 2,E	
212	CALL NC,dddd	SET 2,H	
213	PUSH DE	SET 2,L	
214	SUB dd	SET 2,(HL)	
215	RST 16	SET 2,A	
216	RET C	SET 3,B	
217	EXX	SET 3,C	
218	JP C,dddd	SET 3,D	
219	IN A,dd	SET 3,E	
220	CALL C,dddd	SET 3,H	
221		SET 3,L	
222	SBC A,dd	SET 3,(HL)	
223	RST 24	SET 3,A	
224	RET PO	SET 4,B	
225	POP HL	SET 4,C	
226	JP PO,dddd	SET 4,D	
227	EX (SP),HL	SET 4,E	
228	CALL PO,dddd	SET 4,H	
229	PUSH HL	SET 4,L	
230	AND dd	SET 4,(HL)	
231	RST 32	SET 4,A	
232	RET PE	SET 5,B	
233	JP (HL)	SET 5,C	
234	JP PE,dddd	SET 5,D	
235	EX DE,HL	SET 5,E	
236	CALL PE,dddd	SET 5,H	
237		SET 5,L	
238	XOR N	SET 5,(HL)	
239	RST 40	SET 5,A	
240	RET P	SET 6,B	
241	POP AF	SET 6,C	
242	JP P,dddd	SET 6,D	
243	DI	SET 6,E	
244	CALL P,dddd	SET 6,H	
245	PUSH AF	SET 6,L	
246	OR N	SET 6,(HL)	
247	RST 48	SET 6,A	
248	RET M	SET 7,B	
249	LD SP,HL	SET 7,C	
250	JP M,dddd	SET 7,D	
251	EI	SET 7,E	
252	CALL M,dddd	SET 7,H	
253		SET 7,L	
254	CP dd	SET 7,(HL)	
255	RST 56	SET 7,A	

* INDEXADRESSIERTE BEFEHLE *

Um die Befehle auf das IY-Register zu beziehen, muss 'DD' in 'FD' geaendert werden.

```
DD 09    ADD IX,BC
DD 19    ADD IX,DE
DD 21    LD IX,dddd
DD 22    LD (dddd),IX
DD 23    INC IX
DD 29    ADD IX,IX
DD 2A    LD IX,(dddd)
DD 2B    DEC IX
DD 34    INC (IX+dd)
DD 35    DEC (IX+dd)
DD 36    LD (IX+dd),dd
DD 39    ADD IX,SP
DD 46    LD B,(IX+dd)
DD 4E    LD C,(IX+dd)
DD 56    LD D,(IX+dd)
DD 5E    LD E,(IX+dd)
DD 66    LD H,(IX+dd)
DD 6E    LD L,(IX+dd)
DD 70    LD (IX+dd),B
DD 71    LD (IX+dd),C
DD 72    LD (IX+dd),D
DD 73    LD (IX+dd),E
DD 74    LD (IX+dd),H
DD 75    LD (IX+dd),L
DD 77    LD (IX+dd),A
DD 7E    LD A,(IX+dd)
DD 86    ADD A,(IX+dd)
DD 8E    ADC A,(IX+dd)
DD 96    SUB (IX+dd)
DD 9E    SBC A,(IX+dd)
DD A6    AND (IX+dd)
DD AE    XOR (IX+dd)
DD B6    OR (IX+dd)
DD BE    CP (IX+dd)
```

```
DD CB 06    RLC (IX+dd)
DD CB 0E    RRC (IX+dd)
DD CB 16    RL (IX+dd)
DD CB 1E    RR (IX+dd)
DD CB 26    SLA (IX+dd)
DD CB 2E    SRA (IX+dd)
DD CB 3E    SRL (IX+dd)
DD CB 46    BIT 0,(IX+dd)
DD CB 4E    BIT 1,(IX+dd)
DD CB 56    BIT 2,(IX+dd)
DD CB 5E    BIT 3,(IX+dd)
DD CB 66    BIT 4,(IX+dd)
DD CB 6E    BIT 5,(IX+dd)
DD CB 76    BIT 6,(IX+dd)
DD CB 7E    BIT 7,(IX+dd)
DD CB 86    RES 0,(IX+dd)
DD CB 8E    RES 1,(IX+dd)
DD CB 96    RES 2,(IX+dd)
DD CB 9E    RES 3,(IX+dd)
DD CB A6    RES 4,(IX+dd)
DD CB AE    RES 5,(IX+dd)
DD CB B6    RES 6,(IX+dd)
DD CB BE    RES 7,(IX+dd)
DD CB C6    SET 0,(IX+dd)
DD CB CE    SET 1,(IX+dd)
DD CB D6    SET 2,(IX+dd)
DD CB DE    SET 3,(IX+dd)
DD CB E6    SET 4,(IX+dd)
DD CB EE    SET 5,(IX+dd)
DD CB F6    SET 6,(IX+dd)
DD CB FE    SET 7,(IX+dd)
DD E1    POP IX
DD E3    EX (SP),IX
DD E5    PUSH IX
DD E9    JP (IX)
DD F9    LD SP,IX
```

Quellen

Z80-Assembler-Handbuch, Hofacker.
Ian Logan: Understanding your ZX-Spectrum, Melbourne House.
C.Lorenz: Programmieren in Maschinensprache mit Z80, Hofacker.
mc 1/82, Francis.